PARASOL Developer

User Guide

EXHIBIT D

Title: Business Analysis & Management Systems Utilizing Enterprise Metrics

Inventors: Michael M. Mann & Arne Haugland

Attys.: Fulwider Patton et al. Dkt. # 65568/ENCMP

Parasol Developer User Guide

Object Oriented and Graphical Application Development and Generation

Version 2.4

Developed by Arne Haugland
Written by Jack Figel
Cover Design by Henry Loutsenhizer

© Copyright 1992-1994 Parasol, Inc.
2551 Pebble Beach Loop
Lafayette, CA 94549
Phone: 510-934-8768
Fax: 510-934-8112

First Printing, April 1992
Second Printing, March 1993
Third Printing, October 1994

Limited Warranty

Parasol Inc. warrants that:

- a) The material of the disks and documentation is not defective.
- b) The program is properly recorded on the disks.
- c) The documentation is substantially complete and contains all the information Parasol deems necessary to use the software.
- d) The program functions substantially as described in the documentation.

The sole remedy for breach of this Limited Warranty is the replacement of the defective product. This Limited Warranty is for 90 days from the date of purchase of the software and is extended only to the first user of the software. To replace a defective disk during the warranty term, send the disk and proof of purchase to:

Parasol Inc.
Customer Support Department
2551 Pebble Beach Loop
Lafayette, CA 94549

To replace a disk damaged by the customer during the warranty term, or a disk damaged following the warranty term, send your name and address along with the defective disk and $25.00 for shipping and handling to the above address. The foregoing replacement policy applies only to the original user of the software.

EXCEPT AS PROVIDED ABOVE, AND SUBJECT TO ANY CONTRARY PROVISION OF APPLICABLE LAW, PARASOL INC. DISCLAIMS ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY COVERAGE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES.

Copyright Notice

The manual and the accompanying disks are © Copyright 1992, 1993 by Parasol Inc. All rights reserved. The software described in this document is furnished under a license agreement, and may be used or copied only per the terms of the agreement.

Under copyright laws, this manual or software may not be copied, in whole or in part, without the written consent of Parasol Inc., except in the normal use of the software or to make a backup copy. Under law, copying includes translation into another language or format.

Parasol is a trademark of Parasol Inc. Other trademarks are the properties of their respective companies.

Chapter 1 - Introduction

1.1 Welcome

Welcome to the world of Parasol!

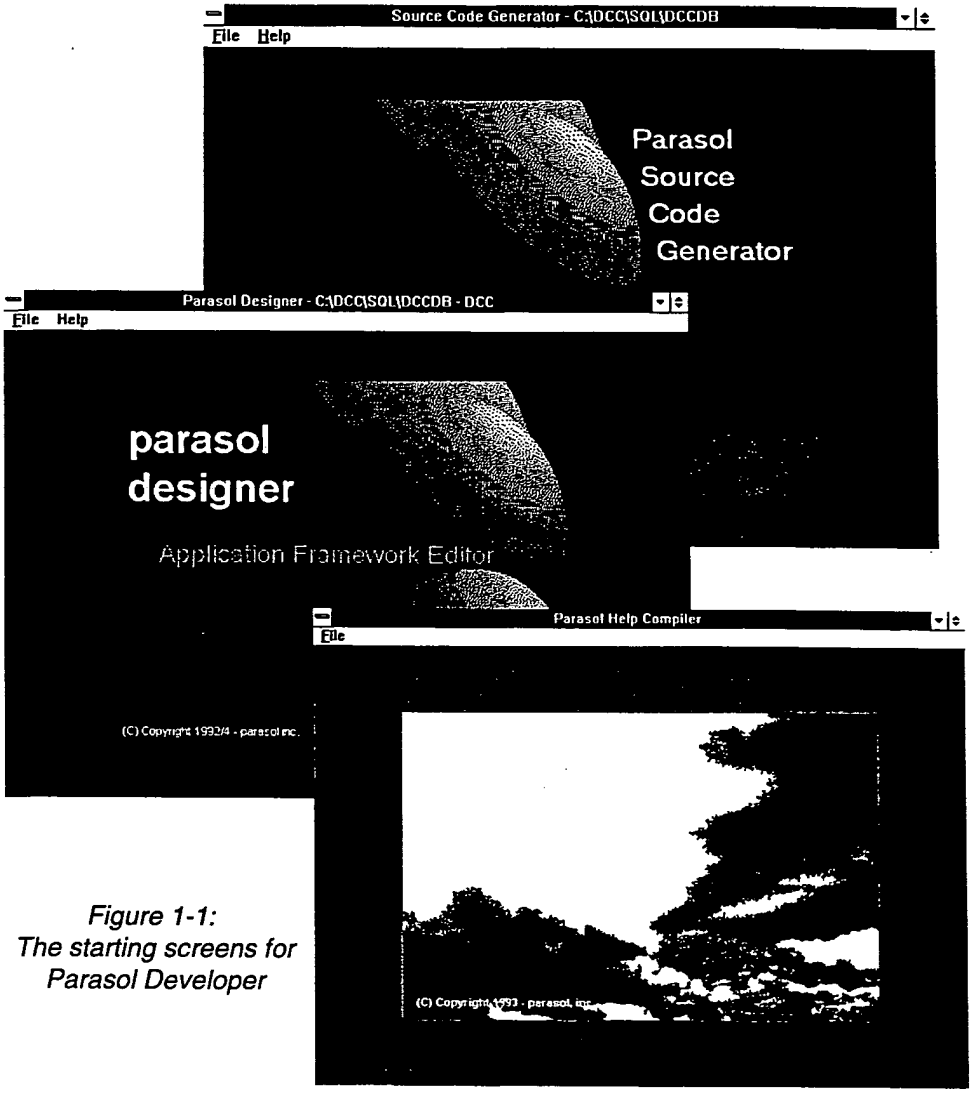

Figure 1-1:
The starting screens for
Parasol Developer

Parasol is a database reference and visualization tool that allows you to relate corporate information together and graphically view information and relationships.

Parasol Developer's major functions are:

- Help you identify and build data-visualization application products
- Design customized applications with object-oriented techniques
- Create the integration of existing databases, or build a database
- Customize the menu options and structure of an application
- Design and layout the dialog screens for your application
- Generate C code that is compiled and linked with Parasol class libraries
- Make a complete, production-quality custom application
- Provide customized help and documentation

By taking advantage of these major features, Parasol based applications provide the following benefits:

- Unravel complex, unrelated data tables for information access
- Make obsolete data tables valuable again without rebuilding the database
- Integrate unrelated applications without restructuring or data transfer
- Save time, development effort and machine costs using existing architecture
- View data graphically to find important information without searching long listings
- Walk through the database structure to investigate relationships
- Comprehend corporate information more completely

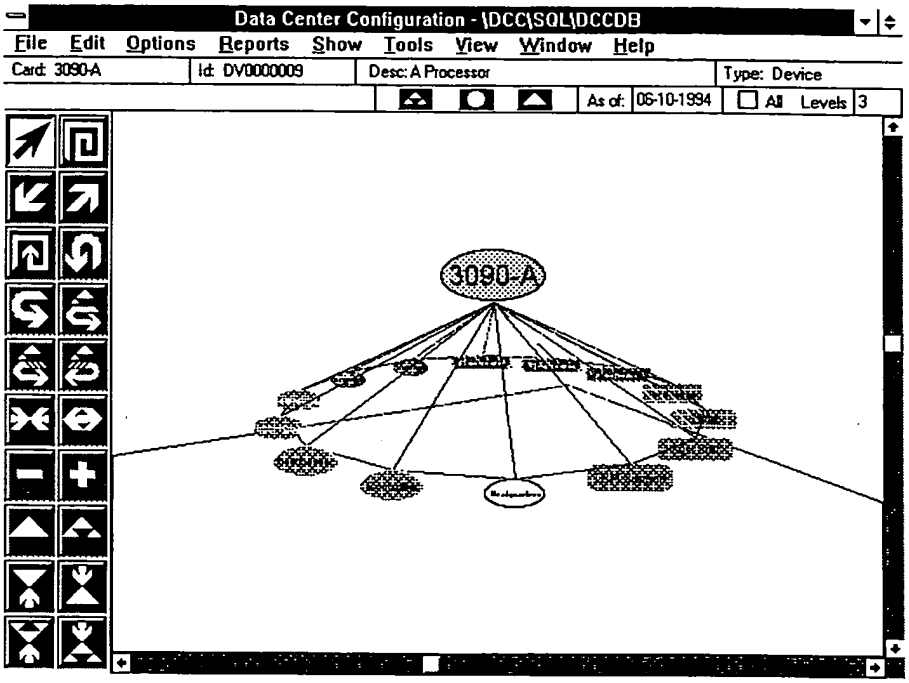

Figure 1-2: Standard graphic view of a database structure in Parasol Executive.

The Parasol product line consists of four separate products. These are:

Parasol Executive - Dynamic Link libraries for applications to function, standard GUI operations, core graphical operations for the end-user, customized menus and dialog boxes.

Parasol Developer - Design and build custom applications, create table definitions, data screens and queries, implement end-user requirements, generate code and compile a production application.

Parasol Applications - Sample applications with ready-to-run data tables, data entry and edit dialog boxes, industry specific functionality, and application specifications in Parasol Developer that can be enhanced or customized for a specific user site.

Parasol Documentation - Electronic form of the user guide for Parasol Executive that can be edited and customized to match a customized application.

1.2 Product Description

The following information describes the major features of Parasol Developer:

Standard Naming Conventions

- Data files, code, and objects utilize standard naming conventions that you can use
- Data tables are defined, either as new tables in an SQL Database, or as existing tables in SQL databases

Customize Menu Structure

- Based on the unique data tables for each application, you can add and/or change the menu names for each user of Parasol Executive
- You can edit other C source code files provided with Parasol Developer to make changes to how Parasol Executive will behave

Design the Application

- Use the Parasol Developer functions to design and build all record queries, table views, find dialogs, report dialogs and data dialogs for the application
- Point-and-click methods and standard graphical icons are used to create object-oriented applications customized for each user

Code Generation

- Run the Parasol Developer - Code Generator to create object-oriented C source code files for each object in the application, and other source code files provided
- Compile the source code into executable libraries and functions

Make the Application

- You can edit any of the source code files to add extra functions, or integrate other applications or modules

- Link the entire application together and merge it into the library modules of Parasol Executive with one execution file
- Install an icon in the graphics environment and attach the application database and executable file for a user to start the customized application

1.3 Default Behavior

Parasol is a Graphical Application Development and SQL Database Visualization tool. It combines this with powerful SQL database editing capability along with audit trails for professional solutions.

Parasol visualizes relationships between rows in tables. Imagine 3 tables drawn on a page.

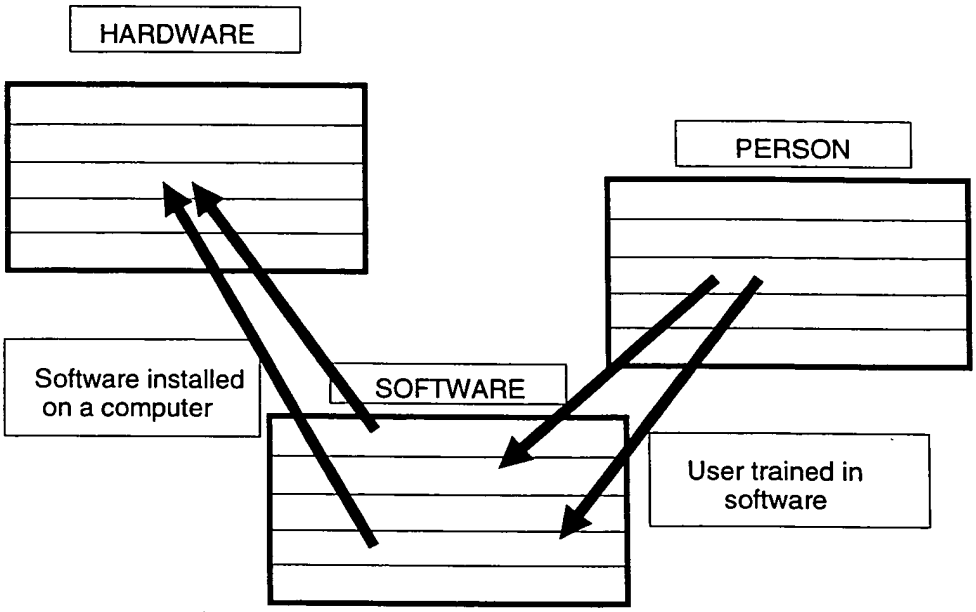

Figure 1-3: Interconnected Tables.

Interconnect any row to any other row on the page, no restrictions other than coming full circle. Parasol will take this data of rows in tables, and their arbitrary connections, and explode or implode a derived heirarchy from a given starting record. Parasol then presents this tree of information as a model in 3D graphic form or an outline

that we can examine and navigate. By pointing at an item we can derive a new heirarchy in either direction. We can walk up or down the tree. The hierarchy can be collapsed and simplified. In 3D view the structure has form, like a model, and we can walk around and into the model.

Parasol allows a user to access the data records, and change the model and/or underlying data. Futhermore, pictures and procedures can be attached to the data rows. Pictures allow for viewing of related data in the form of bitmaps. Procedures allow for communication with other products by passing DDE messages to them.

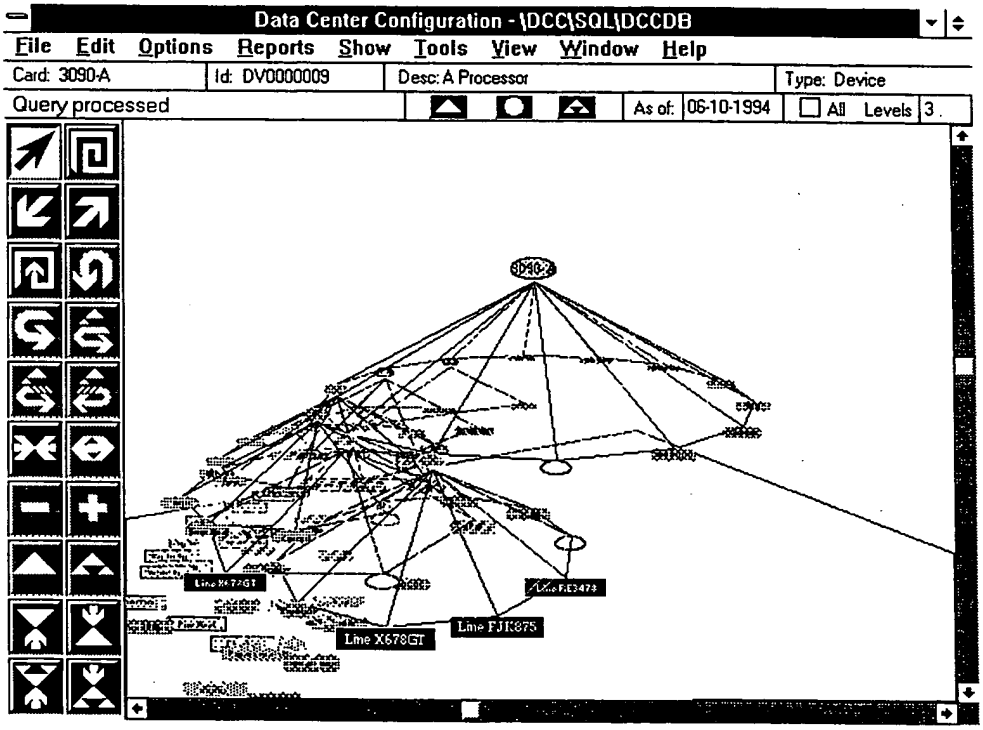

Figure 1-4: Graphic view of a complex structure.

Parasol helps to manage change to the connectivity of the database. A revision record and an audit log are carried with changes about the connections between items in the tables. All connections carry a date when they become effective, and optionally a date for when they become no longer effective. Making connections out effective or deleting them completely is an option you can perform within your Parasol Executive Application.

1.4 Installation

Before you start installing Parasol Developer, make sure you have:

- 486 based Personal Computer (IBM or equivalent)
- Windows Version 3.1 or OS/2 Version 2.1 installed
- SQLServer, ORACLE, OS/2 Extended Services, Gupta SQLBase, WatcomSQL or XDB database installed
- At least 20 MBytes of disk space available
- Floating point processor in your CPU (if you have a 486SX based system)
- 8 MBytes of memory
- For Windows: Microsoft C Compiler C6.0 or C7.0
  Mircosoft Visual C++ 1.5
  Windows Software Development Kit (SDK)
- For OS/2: Microsoft C Compiler C6.0 or IBM C SET/2
  IBM Software Development Kit (SDK)

The installation of Parasol Developer is a simple and straightforward process.

For OS/2 follow these steps:

1) Place the Parasol Developer Installation Disk in drive A or B of your computer.

2) Double click on the A: or B: icon drive on your desktop and you will see the file INSTALL.EXE for Parasol Developer. Double click on it.

3) The Installation Dialog Box will appear. Enter the {drive:\pathname} where you want to install Parasol Developer. And select the checkbox options which are available.

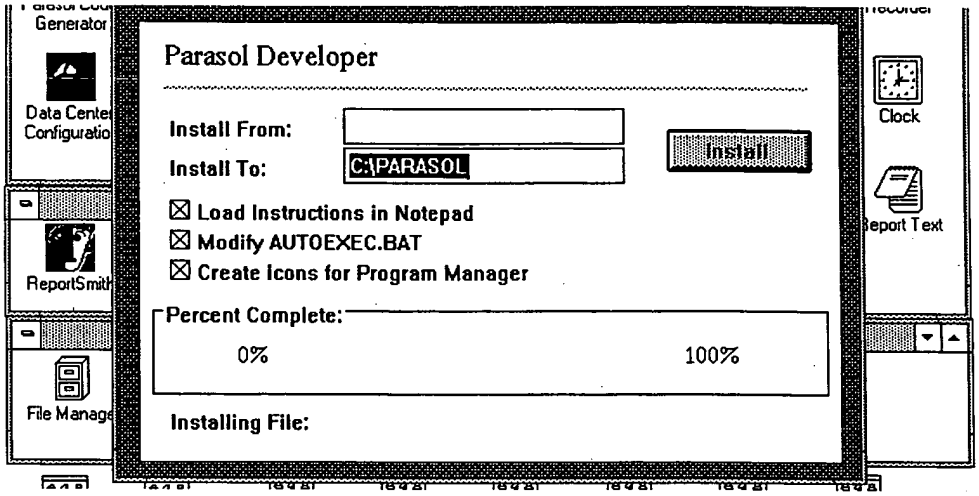

Figure 1-5: Parasol Installation dialog box.

4) The options are:

- Load the instructions file into your System Editor for review
- Modify the CONFIG.SYS file for your system
- Create icons for your desktop

5) Click on the Install button and insert Disks 1 through 2 when requested by the Install Program. You can observe the installation progress by watching the thermometer move toward 100% complete.

6) When the installation is completed, your will automatically be returned to the OS/2 Desktop.

7) If you selected the automatic creation of the Parasol icons, a new Parasol folder will appear with the following:

- Parasol Designer - the Application Framework Editor (Alfred)
- Parasol Code Generator - generates the C code for your application design
- Parasol Help Compiler - converts a text file into an indexed help system

For a Windows installation:

1) Place the Parasol Developer Installation Disk in drive A or B of your computer.

2) Select the Run option of the File menu of Program Manager and enter the name:

A:INSTALLorB:INSTALL

3) This will run the Parasol Developer installation program.

4) In the dialog box that appears, enter the {drive:\pathname} where you want to install Parasol Developer.

5) Select the checkbox options you want to use during installation:

- Load the instructions file into Notepad for review
- Modify the AUTOEXEC.BAT file for your system
- Create icons for Program Manager

6) Click on the Install button and Insert Disks 1 through 3 when requested by the install Program. You can observe the installation progress by watcing the thermometer move toward 100% complete.

7) When the completion message appears, click on the Ok button and the installation process will return your to Program Manager.

8) If you selected the automatic creation of the Parasol icons, a new Parasol group will appear with the following:

- Parasol Designer - the Application Framework Editor (Alfred)
- Parasol Code Generator - generates the C code for your application design
- Parasol Help Compiler - converts a text file into an indexed help system

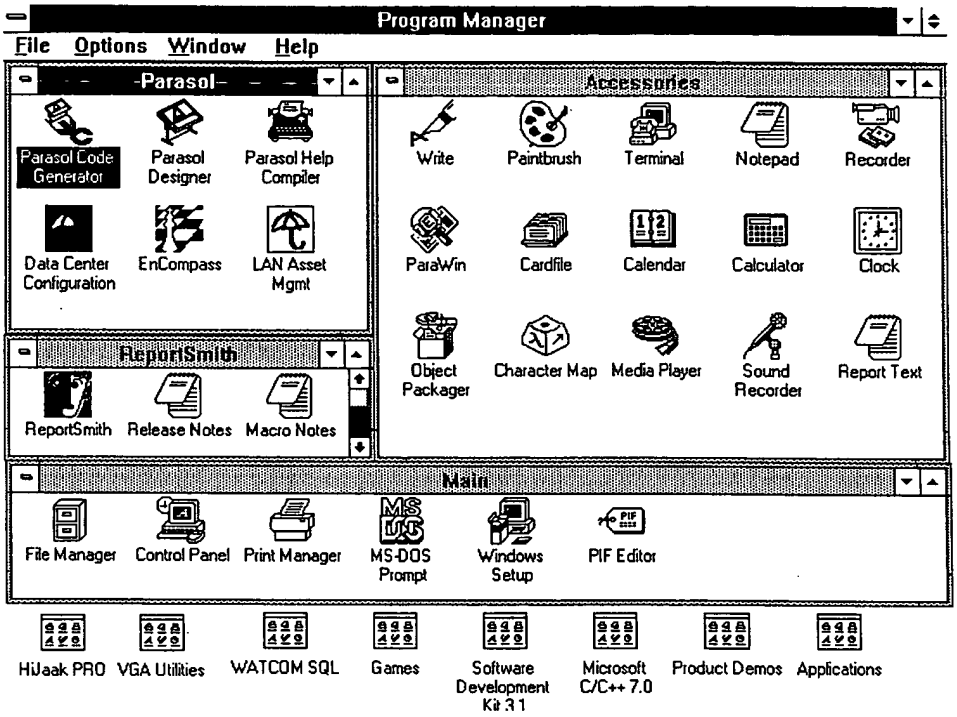

Figure 1-6: Parasol Group with Parasol Developer Installed in Windows.

After you have installed Parasol Developer, the following sub-directories and files were created.

Directories:

- \DLLD - Parasol Development Run-time Dynamic Link Libraries
- \DLL - Parasol Production Run-time Dynamic Link Libraries
- \EXE - Executable programs (Parasol Designer, Parasol Source Code Generator, Bitmap Conversion Utility for OS/2 only, and Help Compiler).
- \BND - Parasol Developer SQL Bind files if required by the database system in use.
- \CPO - C+O Objects. The directory contains the class header files as C header source code.

- \PAR - Parasol Objects. The directory contains the class header files as C header source code.
- \TPL - Parasol Source Code Generator template files.
- \LIBD - Parasol Development Link libraries for linking your application.
- \LIBP - Parasol Production Link libraries for linking your application.

Note that Parasol Developer comes in two forms, Development and Production. You start building your application in development form so that you take advantage of the error checking done as your code is executing. After you are satisfied with your design, and are ready to beta test your application, then you can use the production libraries to generate the executable program.

You will notice that your application's executable program, Dynamic Link Libraries, and Link Libraries are twice the size in development form than in production form. This is because nearly half the executable code is error checking when you link to and run the development libraries.

Also, a Parasol application in development runs approximately 8 times slower than a production version. Always develop using development libraries. When you are ready for beta testing, switch to compiling, linking and running the production libraries.

1.5 User Guide Contents

This User Guide describes how to use the Parasol Developer Product, using Parasol Designer, the Code Generator and Help Compiler. Chapter 1 is an introduction to the major operations of the product, Chapter 2 briefly describes the steps in making an application. Chapters 3-8 give details for each step. The appendices provide detail technical information about how Parasol works and the names of its objects.

Thank you for purchasing Parasol. We hope you enjoy the product, and that it provides the database reference and visualization benefits that you expected.

Questions about the usage of Parasol, product enhancements, or comments about this manual should be submitted to your local distributor, or directly to Parasol Inc.

Chapter 2 - Using Parasol Developer

2.1 Introduction

Parasol is a graphical user interface for SQL databases that is delivered as a set of libraries and classes that you use to build a custom application. As the designer/builder of a Parasol application, your job is to apply this graphical interface to database tables of your own specification. You use Parasol Developer to design and build applications that you run with Parasol Executive. This chapter will briefly describe the steps you take to use Parasol Developer. Then each step is explained in detail in later chapters of this User Guide.

Throughout this User Guide, and in the Parasol Executive User Guide, a simple Local Area Network application example is used to illustrate how the Parasol product works. This product includes a table for **hardware**, **software** and **person**. The diagram below shows these tables in schematic form.

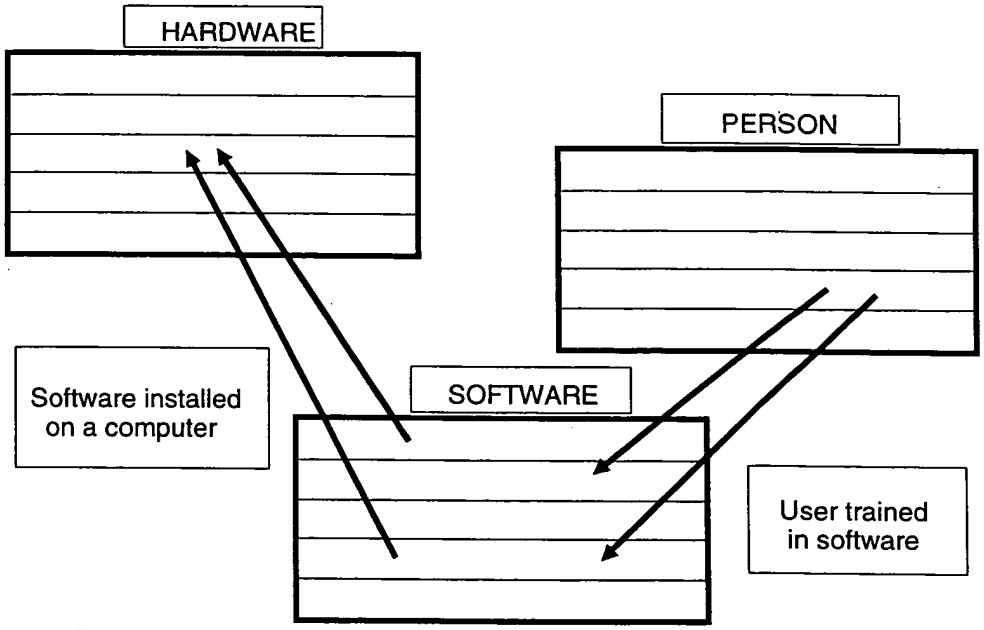

Figure 2-1: Application table design for hardware, software and person.

2.2 Designing an Application

The development process for any application starts with a definition of the database. The first step you should undertake is to design your application's data tables. If they already exist, and you are using Parasol Developer to access them, then you do not need to perform this step.

But if you are building an application from "scatch", then you should layout your database on paper before starting with any work in Parasol Developer.

The design of databases is as much an art as it is a science. You need to understand what kind of information you are going to be entering into your application, and how that data needs to be organized to provide the results you want.

Make a list of the kinds of data you have (e.g. hardware, software and people for the above example). Next, make a list of the pieces of information for each kind of data that you are going to gather and enter into the application (e.g. CPU type, vendor, memory size, disk size, CPU id code, etc. for the hardware table).

The kinds of data you have will be called "tables" in your application, and the pieces of data for each entry in the tables will be called "fields".

You will be designing data entry screens for each table that will contain all the fields for that table. You may want to sketch those out on paper also to refer to later when you design those screens in Parasol Developer.

With the tables, fields, and data entry screens identified and named, you are ready to start building your application.

2.3 Building an Application

Building an application in Parasol Developer is an easy step-by-step process. This process is divided into simple "cookbook" operations to make development easy to perform by both programmers and non-programmers alike.

They are divided into phases as shown in the following chart:

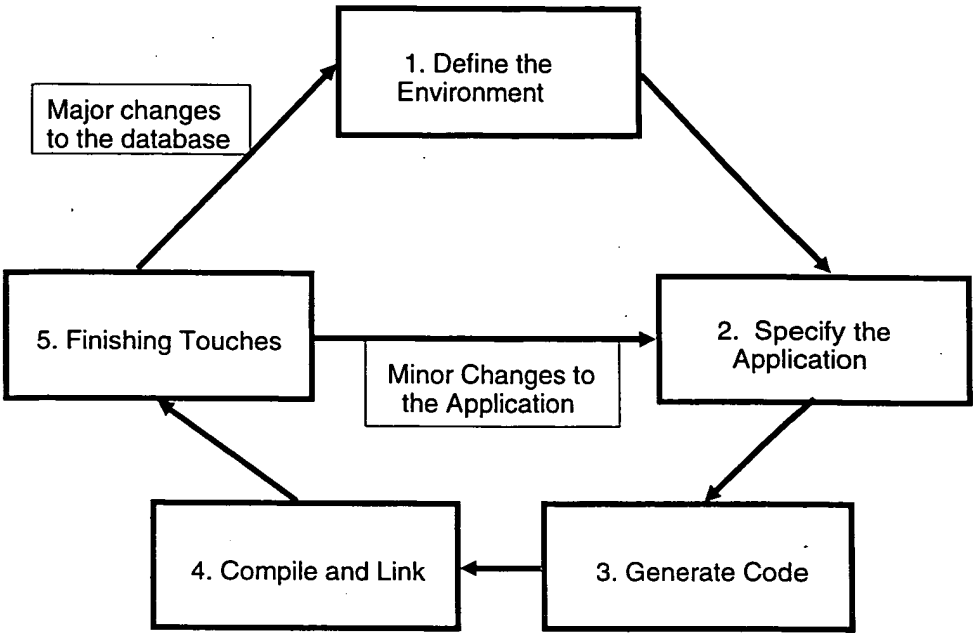

Figure 2-2: Diagram of the application development process.

The phases and steps within each phase are described further below, and in detail in each chapter of this User Guide.

2.3.1 Phase One - Define the Environment

- **Step 1. Create a new application.** You create a working project area with a graphical icon for that application, and run the Create Project function in the Source Code Generator of Parasol Developer. This creates a series of directories and files that you will use as you build your application.
- **Step 2. Create an application database.** In this step, you use the database creation functions of your SQL database system, to create a database in which you will be building the fields, tables and other objects for your application.

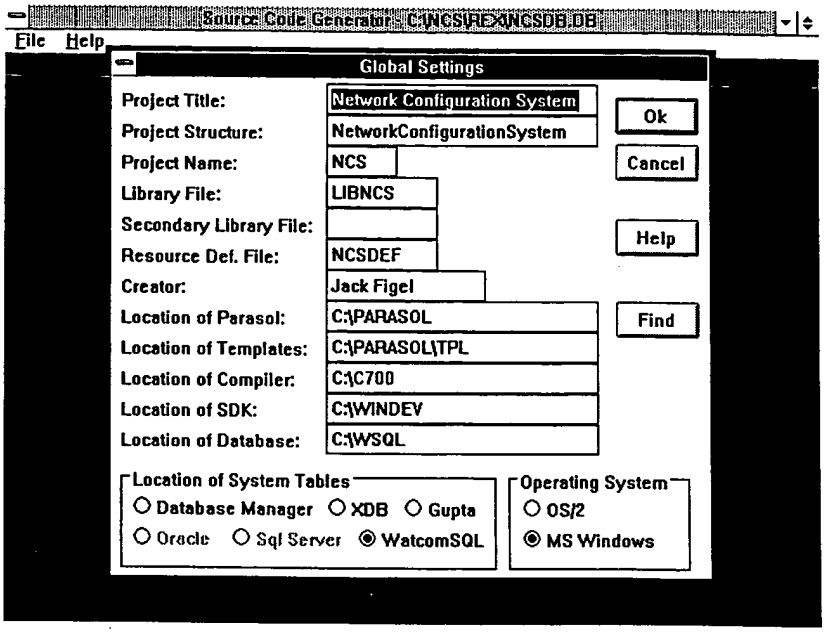

Figure 2-3: Project parameters dialog box in the Source Code Generator.

- **Step 3. Create Parasol system and application database tables.** This step utilizes several SQL language files that are automatically created for you when you start a new application in Parasol. The Parasol system tables are defined together in one database, but your application tables may be distributed among several defined databases, or among various environments (i.e. client/server).

- **Step 4. Establish naming conventions.** Parasol uses a series of naming conventions to keep track of all the objects in your application. It maintains the definitions of objects that are used in your application which are utilized throughout the development process. If you follow the default naming conventions of Parasol, all the object names and definitions are created and maintained automatically.

- **Step 5. Register application tables.** With Parasol Designer, you identify the application tables you have defined in the database, and give them each a 2 character code for quick reference. These code names are the basis for all other naming conventions used throughout your application for menus, dialogs, etc. In this step you also identify the name of the SQL database you are using for each table of your application.

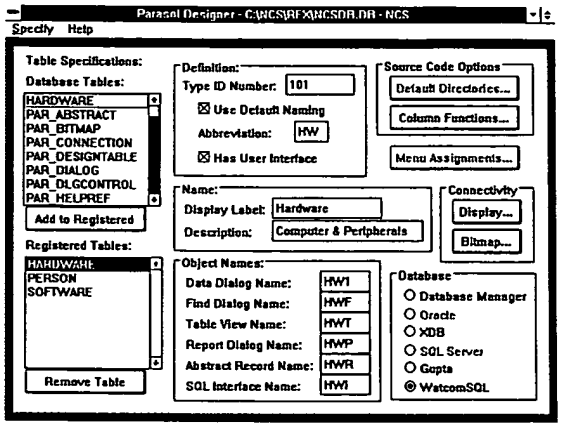

Figure 2-4: Dialog box to register application tables.

2.3.2 Phase Two - Specify the Application.

- **Step 6. Design and build the dialog panels for application tables.** You use Parasol Designer (Application Framework Editor) to define and design the various dialog panels that will appear in your application. You design data entry screens, data table view lists, find dialogs, and report generation dialogs that will be used throughout your application. You use various alignment tools, ordering and tab editing, layouts, and various objects (such as push buttons, entry text boxes, list boxes, etc.) to design your dialogs. The names of objects are automatically assigned when you place them on a dialog panel using graphical "point and click" operations.

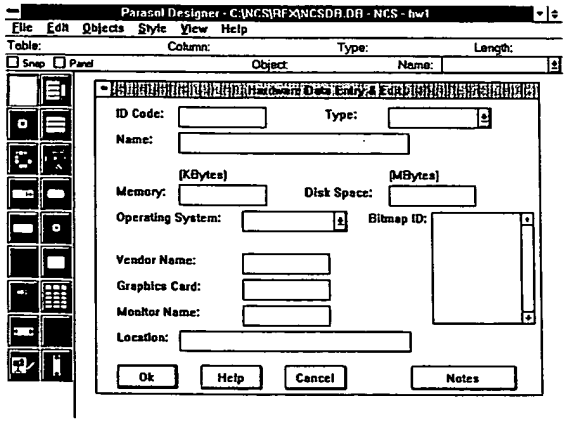

Figure 2-5: An example data dialog box using Layout Tools.

- **Step 7. Link dialog panels to table and table columns.** Once you have designed the dialog panels, you simply link the objects on them to the application tables defined in your SQL database. You use a "drag and drop" graphical technique to quickly and easily connect the dialog panels to the database.

- **Step 8. Design Reports** You use a report writing product (ReportSmith, Query Manager, etc.) to design the content and layout of tabular reports you want in your application. Then you integrate these report definitions with the report dialog panels you have previously defined to produce reports when your application is completed.

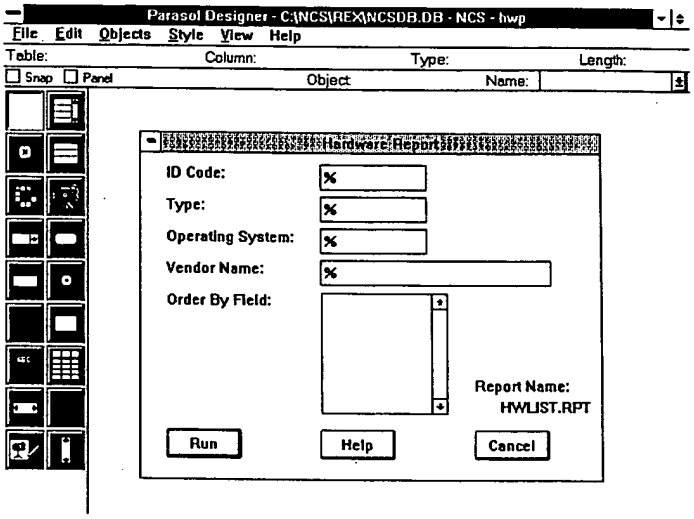

Figure 2-6: Example report dialog box for a Parasol based application.

2.3.3 Phase Three - Generate Code

All the steps below are automatically performed when you select the menu option to **Generate All** objects in your application. These are described separately to demonstrate the comprehensive functions of the Source Code Generator.

- **Step 9. Generate "C" code for the Application Objects.** These are the application object, menu manager object, record manager object, and the tree building object. These objects are required to make your Parasol application operate in its default behavior modes, including the menus, records, and methods for building data table relationships.

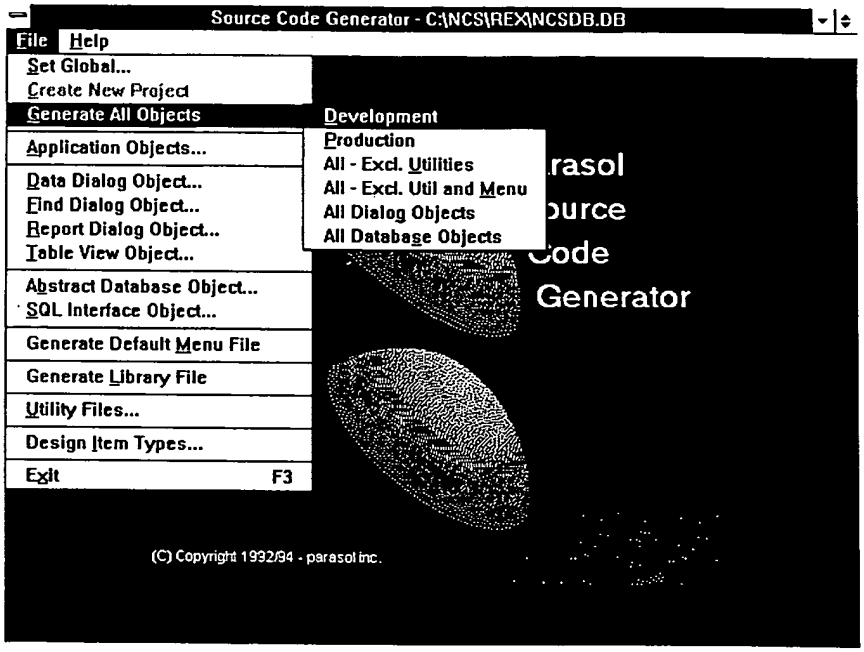

Figure 2-7: Main menu for Parasol Source Code Generator.

- **Step 10. Generate "C" code for the data dialog objects, find dialog objects, and the table view objects.** These objects were specified by Parasol Designer in the previous steps. The code generated is based on the fields on the panels, the control options for each field type, and other functions such as the Ok, Cancel and Help buttons.

- **Step 11 Generate "C" code for report dialog objects.** These objects were specified by Parasol Designer and represent the dialogs that will be used to generate reports in the application. The code generated is based on the parameters you specified on the panels, and the reports that are associated with each panel.

- **Step 12. Generate "C" code for the abstract database record objects and the SQL interface objects.** These objects tie the definition of your application objects to the actual SQL database you are using for your application.

- **Step 13. Generate the Menu File.** A special menu option of the Source Code Generator will create a default menu file for all the data tables, dialog panels, etc. you have defined for your application. You can then edit this file to remove, change or add any functionality for your specific application.

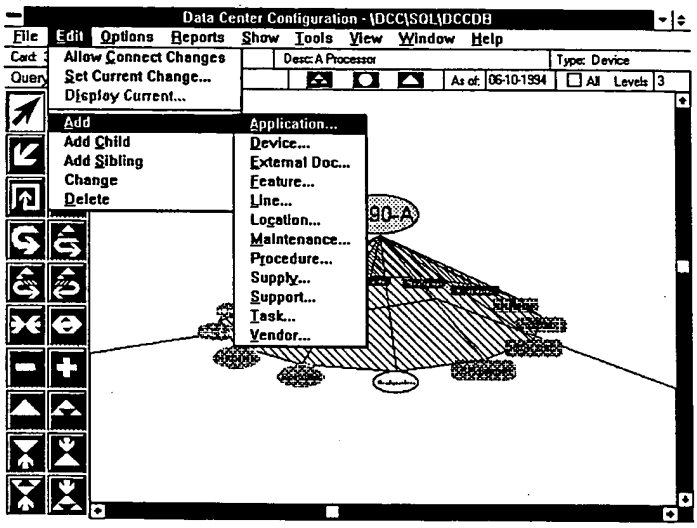

Figure 2-8: A typical menu structure in a Parasol application.

2.3.4 Phase Four - Compile and Link

- **Step 14. Insert an application logo as a bitmap, customize the icon and accelerator table.** As a final step before compiling the entire application, this step shows you how to change the bitmap logo for the startup screen of your application. Also, you can customize the icon that appears in Windows or OS/2, and make modifications or new entries in the accelerator table for short-cut keystrokes that will be valid in your application.

- **Step 15. Compile and link the generated code.** In this step, you first generate the "make" and "link" command files using the Source Code Generator. You then run these command files from the operating system to compile all the objects you have designed and generated, link the resulting executable code together, and integrate them with the Parasol libraries to create a production application.

- **Step 16. Load application specific data into system tables.** If you have specific application data that needs to be entered, or loaded, into your database, you can do this at this point. Usually this will consist of validation tables or selection lists for list boxes, combo boxes, and any changes you may want to make to the valid options for other system values.

2.3.5 Phase Five - Finishing Touches

- **Step 17. Configure the application.** For each user and/or workstation, you can specify the colors, fonts, layouts, valid menu options, choice lists, additional security (above what the database provides), filters, startup "look and feel", and much more. This is achieved by giving each user a private profile file combined with data in system tables.

- **Step 18. Add context sensitive help.** For each dialog panel, you can identify context sensitive help when a user selects the Help button on the dialog panel. This step explains how to tie in your own help system for your application.

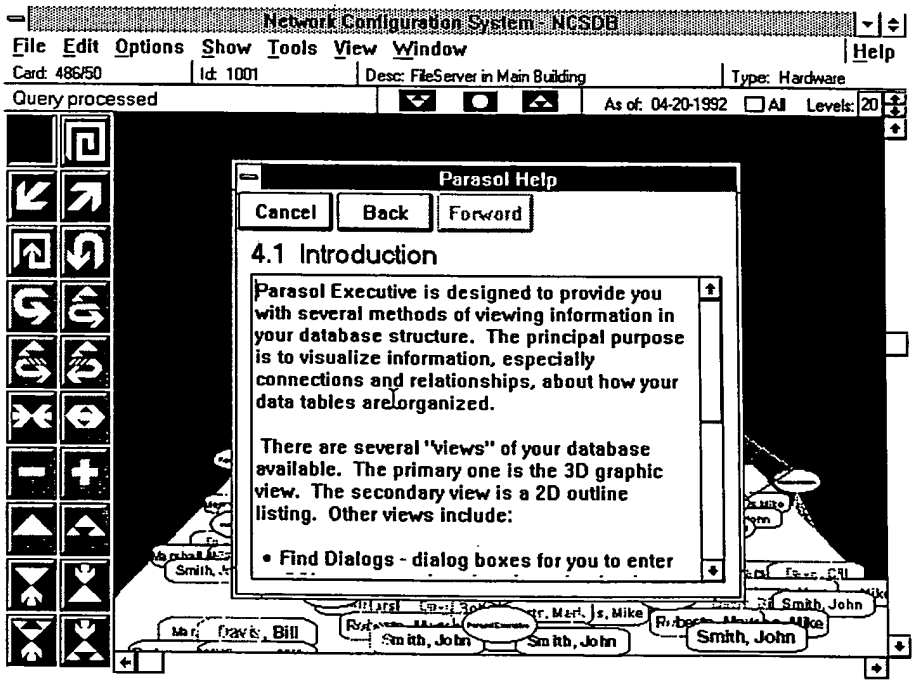

Figure 2-10: Example of a help screen from a Parasol generated application.

- **Step 19. Place all Dialogs into a Dynamic Link Library.** To improve performance, you move the application window designs from the database to Dynamic Link Libraries, and integrate them with other production libraries of Parasol to create a final, efficient application program.

- **Step 20. Create an installation procedure.** In this step, information is provided to assist you in creating an installation procedure for your application using the Parasol Installation program. This makes it easy to distribute your application to the users in your organization, without extensive technical assistance for installing their copy.

When you have completed these steps, you will have built a ready-to-run application that runs with the Parasol Executive libraries. Refer to chapters 3-8 of this User Guide to find further information about each step, and the Parasol Executive User Guide on how an application operates when it is completed.

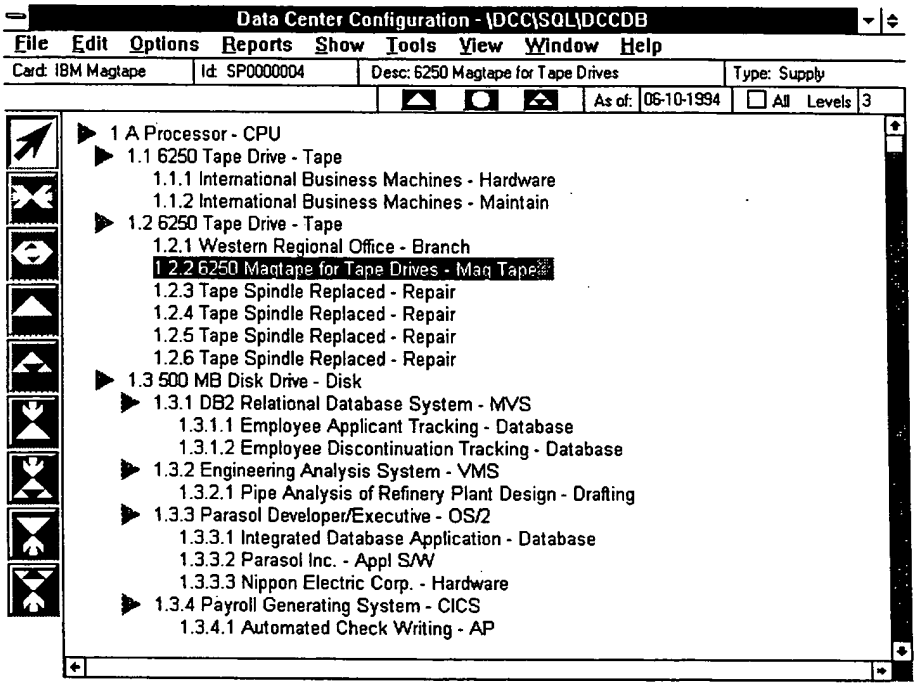

Figure 2-11: Typical 2D outline view of data in a Parasol application

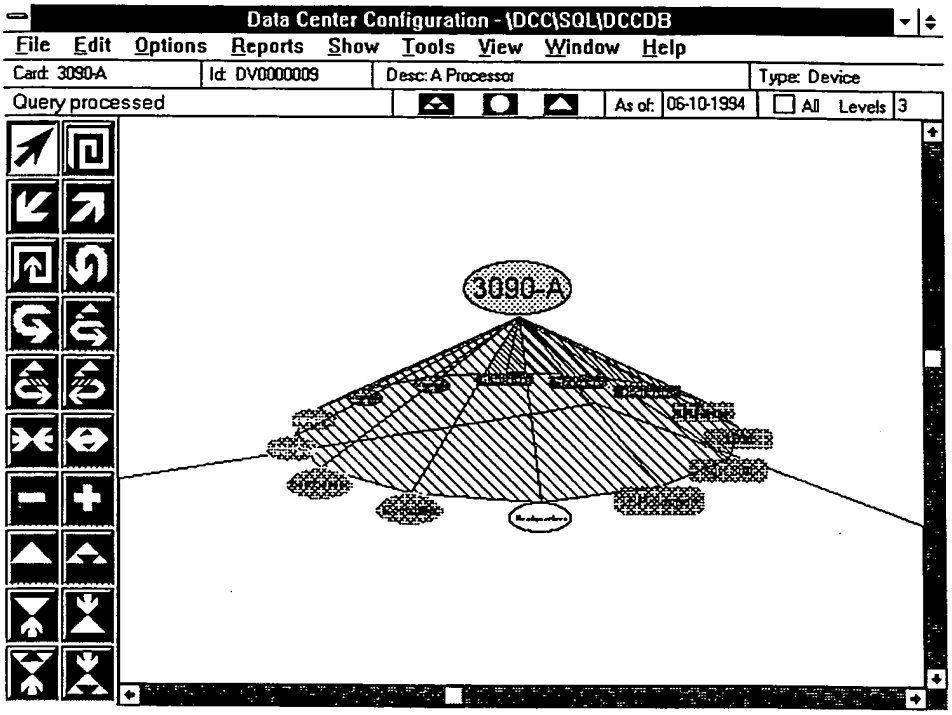

Figure 2-12: Typical 3D graphic screen from a Parasol application.

2.4 Application Objects

Parasol Developer and Parasol Executive employ object-oriented design, development and implementation techniques. These are today's current methods for building efficient, re-usable programs for windowing environments. There are several classes of objects in a Parasol application. When you specify your application you will create application objects such as dialogs, table views, and SQL database covers.

Parasol Developer creates the following system level objects which make your application function:

AMM - Application menu manager object.

The procedures in this file are called when the user touches your menu item. Here you can accept the default behavior or not.

- AmmFind: Find record
- AmmAdd: Add record
- AmmChild: Add child record
- AmmSibling: Add sibling record
- AmmChange: Change a record
- AmmDelete: Delete a record
- AmmRunReport: Run a report
- AmmGreyChangeRecord: Should change record menu be greyed
- AmmGreyDeleteRecord: Should delete record menu be greyed

XYZ (project name) - Application object

This is the object that inherits the Parasol application object and will be named from the three character project name for your application. The application object is the program start entry point when your application executes. The various command files that are created for your application will have the project name imbedded for automatically you. The project name is established in the Parasol Source Code Generator, Set Global menu command.

ARM - Application record manager

This object holds one global variable for each table type in your application. The variable is one instance of an abstract record of the database table.

TBF - Application tree building callbacks

These methods give you control of the display in outline or 3D.

- TbfGetDescriptionLine: Sets the display
- TbfGetPictureCaption: Sets the picture caption

- TbfGetDisplayItemType Sets display design record to use
- TbfIsItemToBeDisplayed Include record in display

These objects are generated by the Parasol Source Code Generator and give you access to making major changes in the default behavior of a Parasol Executive application.

2.5 Resource Objects

In addition to system objects, a Parasol application also consists of resource objects. These include system related objects and application specific objects.

The system level objects are:

- A logo bitmap
- A menu
- An icon
- An accelerator table
- Various mouse pointers (Windows only)

These resources, except for the application icon and mouse pointers, are stored in a Dynamic Link Library (XYZRES.DLL where xyz is the name of your project). The application icon is compiled into the application executable file. Other resources can be added to the library and used in dialog boxes.

The major purpose of Parasol Developer, and at the discretion of the designer, is the design of the dialog panels that appear in the Parasol application. Your job is to present a Parasol application user with four dialog panel types for each table in the database. Tables may also be defined that have no user interface in a Parasol application.

2.5.1 Data Entry Screen(s)

There is at least one data entry dialog box for each table. This can be a single dialog box or, if the table is large enough, it can be several dialog boxes chained together as the designer wishes. The dialog box(s) should provide access to all the columns in a table that are being used. Up to seven "satellite", dialog boxes can be defined for each table.

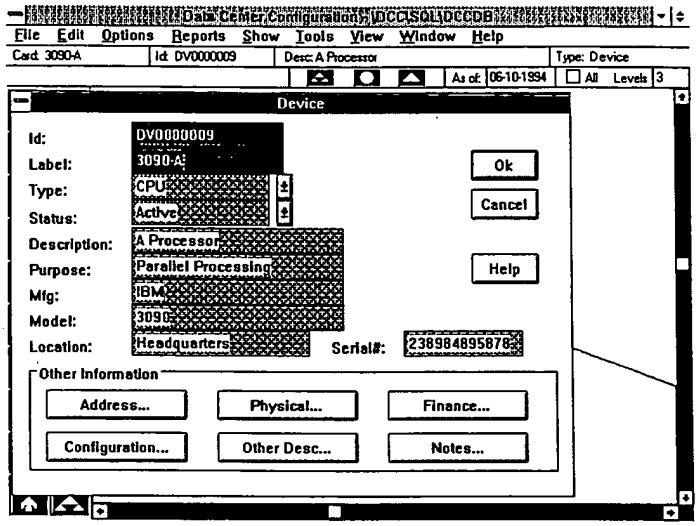

Figure 2-13: Typical Data Entry Screen.

2.5.2 Query Screen

A dialog box, called the "find" dialog, allows the user to do an SQL query against a table with an SQL where clause. This dialog will execute the SQL statement to derive data rows that are display in the Table View of Parasol. You can also define several different find dialog boxes for each table.

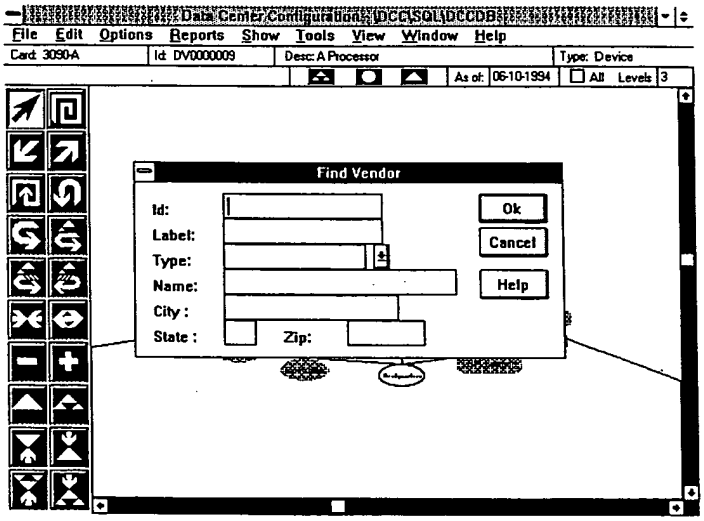

Figure 2-14: Typical Query Screen (Find Dialog Box).

2.5.3 Table View

There is a Parasol Table View for each table in the database of your application. The table view is a row and column view of data in a table. This listing appears as a result of using the find dialog box. The table view then allows for selection of a row for starting an explosion or implosion from this record. A query can be resubmitted, and a different set of rows can be viewed. Data records for that table can be added (unconnected that is), changed and deleted. Records that are not connected to any other record (orphans) can also be queried.

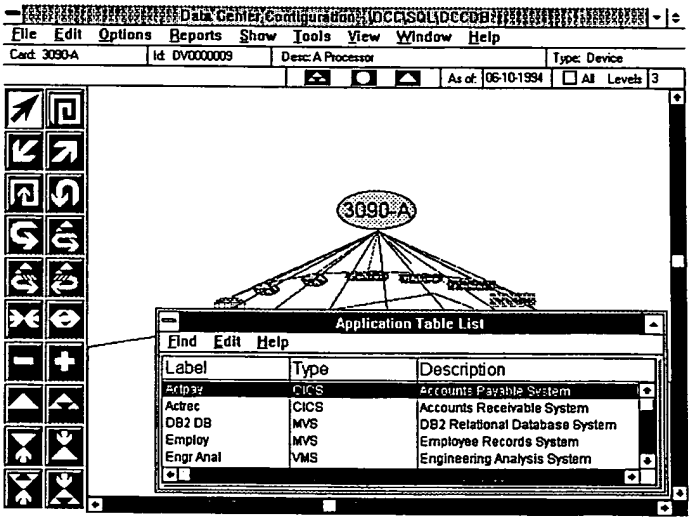

Figure 2-15: Typical Table View Screen.

2.5.4 Report Screen

For each table in the database a user should be able to run a report or a series of reports that present all the data in the table. Reports are basically an SQL statement and a format definition to present the results of a query. SQL statements in report writers allow for variables so a user can select on the data (what column to order by is a typical example). The Parasol Report dialog gets the selection criteria from the user and passes these to the report.

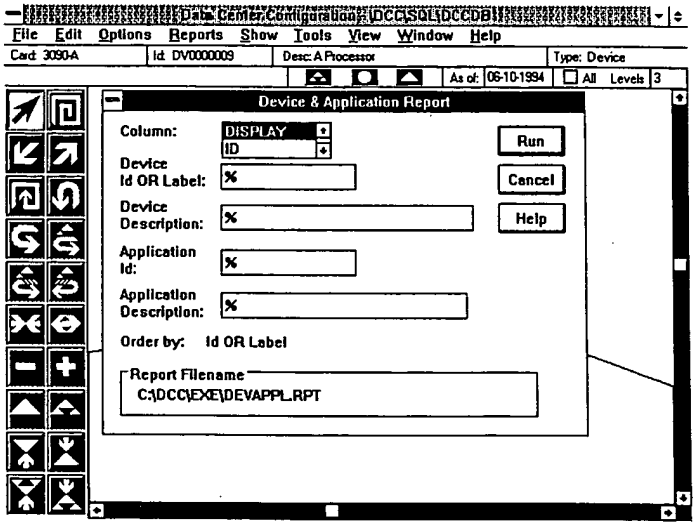

Figure 2-16: Typical Report Screen Screen.

2.5.5 Integration of Objects

As the application designer and builder you have the discretion to use as many or as few of these objects as you want to suit your specific requirements.

Each one of the above dialogs and tables is represented as one "C" object. These objects are generated using the Parasol Source Code Generator. The specification for these objects is stored in the database using Parasol Designer (Application Framework Editor). Once the specification is complete, you run the Parasol Source Code Generator to create an object, or replace an existing object.

The generated dialog/table objects communicate with the database through the Abstract Database Record Object, which in turn communicates through an SQL Interface Database Object to the actual SQL table. Both these objects are generated by the Parasol Source Code Generator. The SQL Interface Database Object contains the native database calls. The dialog/table objects know only about the Abstract Database Record Object, which basically consists of SET and GET column calls as well as FETCH, INSERT, UPDATE and DELETE row calls. This makes it simple to introduce specialized code that accesses the tables, and to change the default behavior of a Parasol application.

You can also generate an Abstract Database Record Object and an SQL Interface Object for tables that have no user interface in a Parasol application, and thereby access database records from your application for specialized behavior.

All the above objects are compiled using the MircoSoft C Compiler or the IBM C SET/2 Compiler, and linked to the Parasol libraries. The resulting executable file is the final application product.

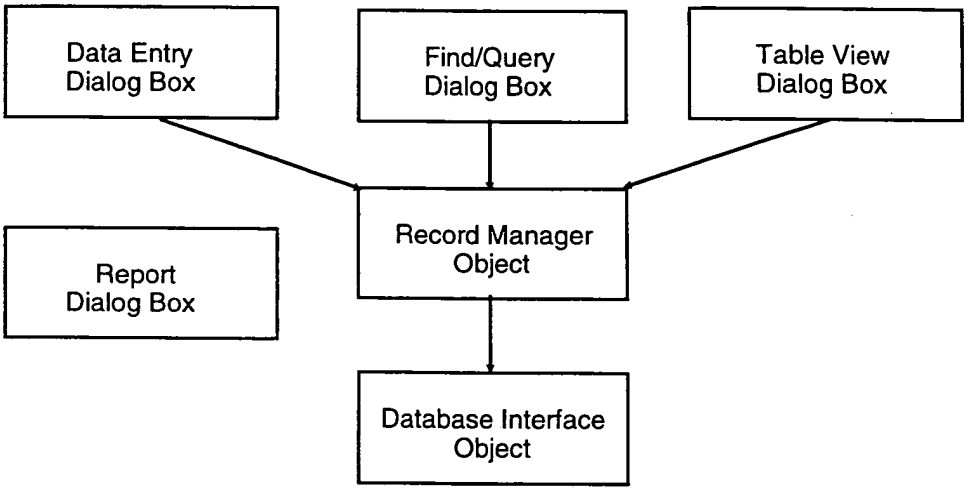

Figure 2-17: Link between dialog panels, abstract database record and SQL interface objects.

Chapter 3 - Define Environment

3.1 Create Windows Group or OS/2 Folder for Project

This is step 1 in creating a new Parasol application.

Create an empty group/folder and then copy into this group/folder each of the three Parasol Developer icons from the main Parasol group/folder.

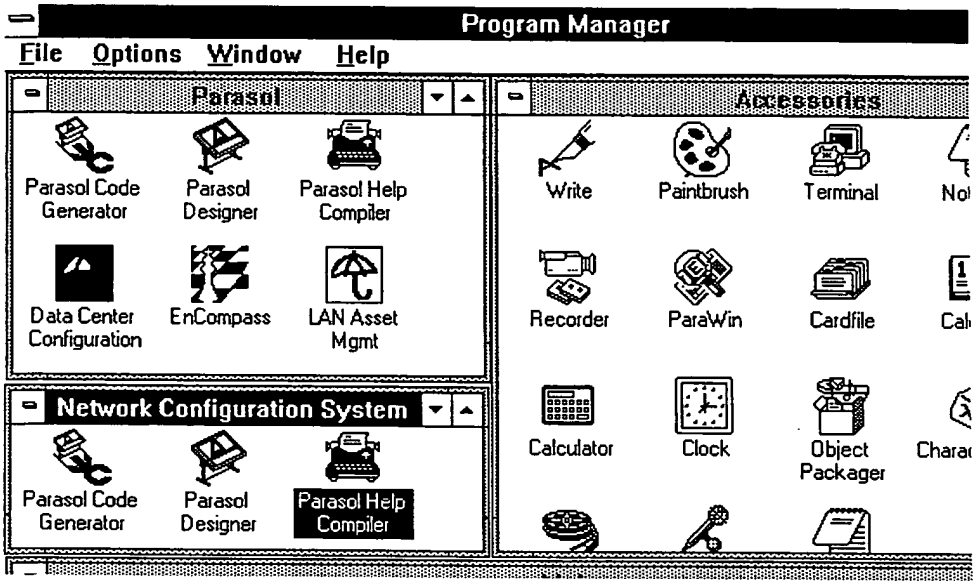

Figure 3-1: New Program Group for the Network Configuration System.

3.1.1 Create New Project

Select with a single click (not a double click) the Parasol Source Code Generator icon in the Parasol Group or Folder you created above. For OS/2 select the Object Settings for that icon from the menu options. For Windows, select the Properties option of the File menu when the icon is selected.

Ensure that the icon runs with no parameters and set the working directory to the root of the drive where you want the new project to be located. Then exit this dialog and save the new parameters.

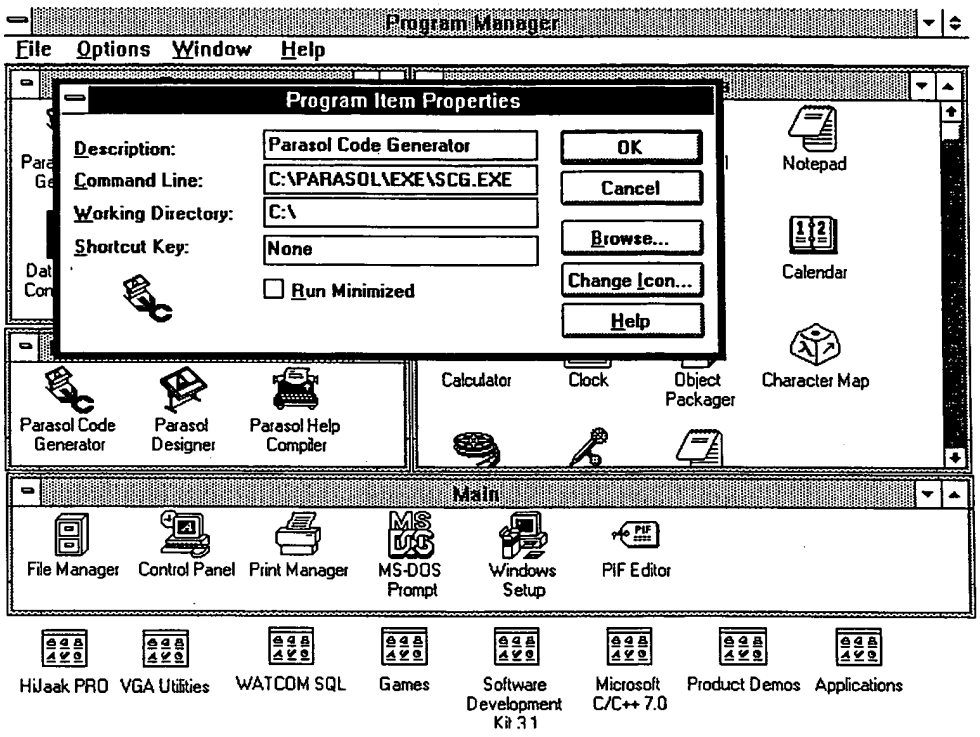

Figure 3-2: Properties dialog box for Parasol Source Code Generator

Parasol Developer uses a three letter naming convention for projects and objects. So, invent a 3 character name for your project. However, be careful, there are about 400 reserved names in Parasol Developer that you cannot use. The object names that already are used by Parasol Developer can be found in Appendix C.

The easy way to query reserved names is to do an DOS or OS/2 directory command on the object names in the Parasol Developer directories \PAR and \CPO. Refer to Appendix B for details on the naming conventions used for "C" objects.

Open the Parasol Source Code Generator by double clicking on its icon. This will run an executable file named SCG.EXE and allow you to initialize several project parameters for your new application.

3.1.2 Enter Global Settings

Select the menu option Set Global in the Parasol Source Code Generator to enter information about your project.

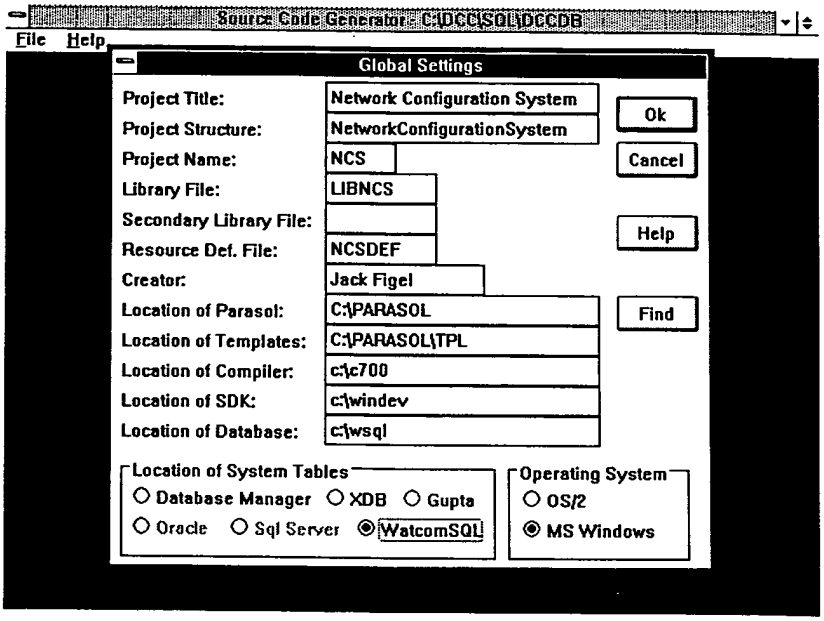

Figure 3-3: Global Settings dialog box for a new project.

Fill in the following:

- Project Title: - The name that appears on the application title bar.
- Project Structure: - The name of the C structure that represents the top most object of your project. It must be more than 3 characters long. You can name it the same as the project title (spaces will be removed).
- Project Name: - All project names consist of three characters that do not conflict with any of the reserved names detailed in Appendix C.
- Library File: - The name of the file that defines the names of "C" objects. This file is automatically maintained by Parasol Developer. A file name of LIBXYZ, where xyz is the name of your project, is assumed and is automatically created for you later. A file extension of .H is assumed and must not be included here.

- Secondary Library File: - The name of the secondary file that defines the names of "C" objects. This file is manually maintained by the developer. Unless you are inserting your own "C" objects into a Dynamic Link Library this field should be blank.
- Resource Def. File: - The name of the file that defines your naming convention for dialog controls, menu items, dialog panels, bitmaps and icons. This file is automatically maintained by Parasol Developer. A file name of XYZDEF, where xyz is the name of your project, is assumed and is automatically created for you later. A file extension of .RH is assumed and must not be included here. @BULLETED LIST = Creator: - The name of the developer. This name will be inserted to all source code that is generated.
- Location of Templates: - This is the location on the disk drive of the Parasol Source Code Generator template files. The Parasol Installation Program places the template files in the directory \PARASOL\TPL.
- Location of Compiler: - This is the location on the disk drive of the compiler you will be using.
- Location of SDK: - Location on the disk drive of the Windows or OS/2 SDK.
- Location of Database: - Location on the disk drive of the database software programs you will be using.
- Location of System Tables - This is the database of choice where the Parasol system tables will be located. Parasol requires that the system tables be located in ONE of a selection of databases.
- Operating System - Indicate whether the project is for Windows or OS/2. Note that the only files of a Parasol application that are affected by operating system are resource related, environmental setup, and link.

These variables will be saved in the profile file SCG.INI in the working directory which should currently be the root of the drive where your project is to be located.

3.1.3 Create Project

From the File menu in the Parasol Source Code Generator, select the option Create Project. This will build all the project environment objects necessary to building your application based on the global parameters you just entered.

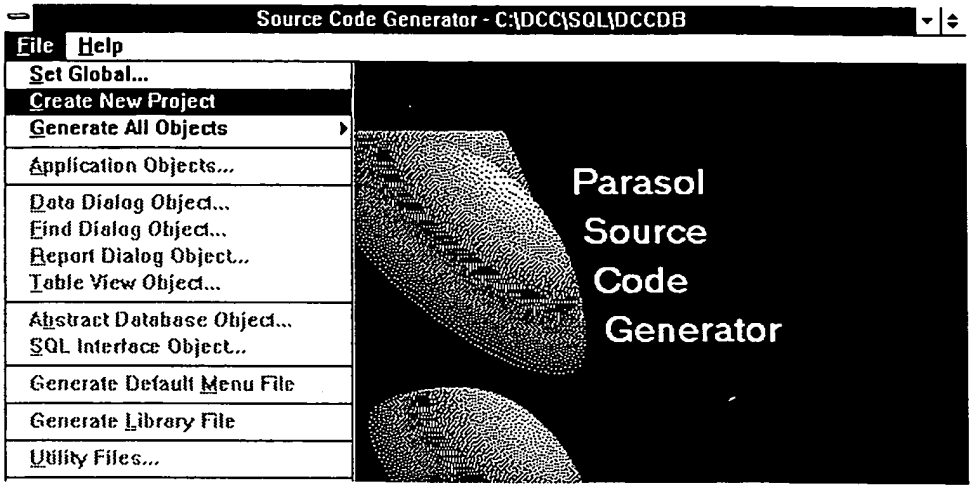

Figure 3-4: File Menu with Create Project option highlighted.

The Parasol Source Code Generator creates projects on the current disk drive identified by the working directory of the Source Code Generator icon settings.

The following directories are created from this process, where xyz is the name of the project:

- \XYZ Project directory
- \XYZ\CL Command log directory. Holds error log output from the make utility.
- \XYZ\CMD Command file directory. Holds OS/2 and Windows command files.
- \XYZ\DSC Development source C files.
- \XYZ\DSH Development source H files.
- \XYZ\DSI Development source Icon files.

- \XYZ\DSO Development source Make files.
- \XYZ\DSR Development source Resource files.
- \XYZ\EXE Executable files, Dynamic Link Libraries and application SQL Bind files. Output from the Linker.
- \XYZ\OBJ Object files. Output from C compiler.
- \XYZ\REX Application Procedural SQL files and data import/export files

When this process is complete, exit the Parasol Source Code Generator at this stage to perform other functions.

The Parasol Source Code Generator creates a personal profile file named SCG.INI. This profile file contains the configuration of the Parasol Code Generator the last time it was run. When the project is created this file is copied from the root drive (the current working directory) into the directory \XYZ\REX where XYZ is the name of the project you created. In order for this file to be found each time you run the Parasol Code Generator, always run the Source Code Generator program from an icon where you can specify the working directory. Change the working directory of your icon to \XYZ\REX where XYZ is your project name.

3.2 Create Application Database

3.2.1 Create a Database

Create a new database in the SQL system you are using. The database should be placed in the directory \XYZ\REX where XYZ is your proejct name. Information about creating a database is included in a README file for Parasol for the various databases which are supported, or can be found in the documentation for the database system you are using. When creating the database, or creating tables in a database, the database administrator user name must be "Parasol". Before attempting to use Parasol Developer, you must create a user with database administrator priviledges name "Parasol". Use this database signon name for all Parasol Developer tasks.

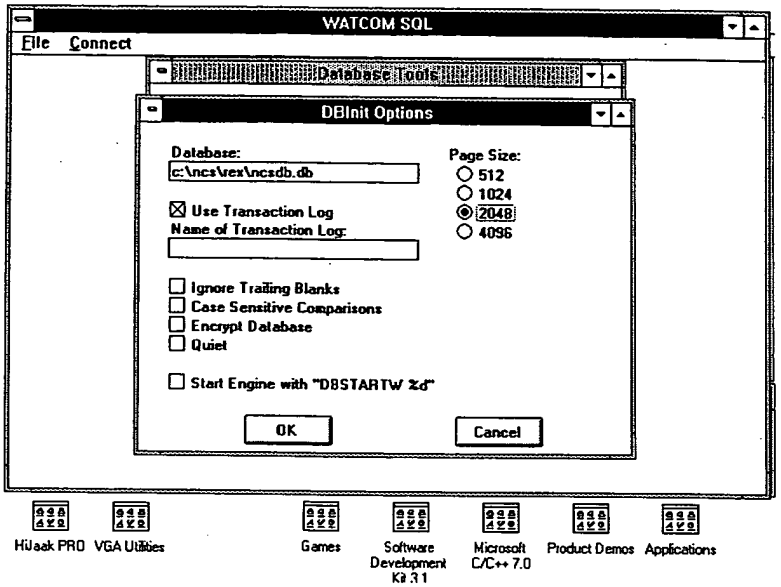

Figure 3-5: Database initilization options in Watcom SQL for Windows.

If you name the database XYZDB, where xyz is the name of your project, you will not need to make any file changes. However, if you use some other naming convention, it is not difficult to make the edits later in the application generation files.

Refer to your database system documentation for further details on making a database.

3.3 Application Tables

When you created the project with the Parasol Source Code Generator, it created a number of Procedural SQL files in your \REX subdirectory of your project directory XYZ. These procedures will build a Parasol database. The next step is to edit these existing files with information about your database design that can automatically establish data fields and tables for your application.

3.3.1 Edit Command Files

Using File Manager or the OS/2 disk icons view the REX directory and you will find all the following files located there. The XYZ will be replaced by your project name. For Database Manager the abc will be replaced by CMD, for XDB or WatcomSql it will be SQL, and for Gupta SQLBase it will be WTS. The files created are:

- XYZCRTB.abc - Create Parasol system tables. See Appendix A for details on the Parasol System tables.
- XYZCDTB.abc - Create Parasol Developer system tables. See Appendix A for details on the Parasol Developer system tables.
- XYZCATB.abc - Create application tables. This is a blank procedural SQL file for inserting your own CREATE TABLE statements.
- XYZLOAD.abc - Load data into the PAR_SELECTION table, the PAR_ITEMTYPE table, and the PAR_PARCONTROL table. This data must be loaded.
- XYZDATA.abc - Load sample data into the PAR_ABSTRACT table, the PAR_ITEM table, the PAR_CONNECTION table, and the PAR_REVISION table. This data shows a connected structure where all the data resides in the PAR_ABSTRACT table. This data is optional. (for XDB this file is divided into two data files named XYZDATA1 and XYZDATA2).
- XYZHELP.abc - Loads references for on-line help system. This data must be loaded. In this file you insert your references from dialogs to your own help file.
- XYZCRIDX.abc - Create indices on Parasol system tables. This is not optional for performance reasons. Here you can create your own indices.
- XYZBIND.abc will bind each Parasol system table to the database. Note that this is only required for Database Manager.

For Database Manager and WatcomSql:

When you execute XYZCREAT.abc, where xyz is the name of the project, this will run the all the command files listed above to create your data tables in the database.

For XDB and SqlBase databases:

You must use the standard query language mode for the database you are using and execute each of the files listed on the previous page in sequence. These files will create the system datatables and load basic Parasol system information.

3.3.2 Edit Table Definition File

Edit the Procedural SQL file XYZCATB.abc where xyz is the name of the project and abc varies by database. For Database Manager, abc is CMD, for Gupta SQLBase abc is WTS, and for WatcomSql and XDB it is SQL. Add the SQL statements to create your application tables. The file contains a sample table that shows the syntax of Procedural SQL.

The information below shows a sample table created in Procedural SQL:

```
DROP TABLE PARASOL.SAMPLE

        CREATE TABLE PARASOL.SAMPLE (
        ID                  CHAR(15)      NOT NULL,
        TYPE                CHAR(15)      NOT NULL,
        VALUE               SMALLINT,
        NUMBER              INTEGER,
        DESCRIPTION         CHAR(50),
        NOTES          VARCHAR(100),
        REQDATE             DATE,
        PLNSTARTTIME        TIME,
        INSTALLED           TIMESTAMP,
        VERSION             DECIMAL(3,1);
```

Figure 3-6: Sample procedural SQL.

Each table in your application should have the DROP TABLE statement and the CREATE TABLE statement.

The following shows an example file for creating tables in the Network Configuration System application:

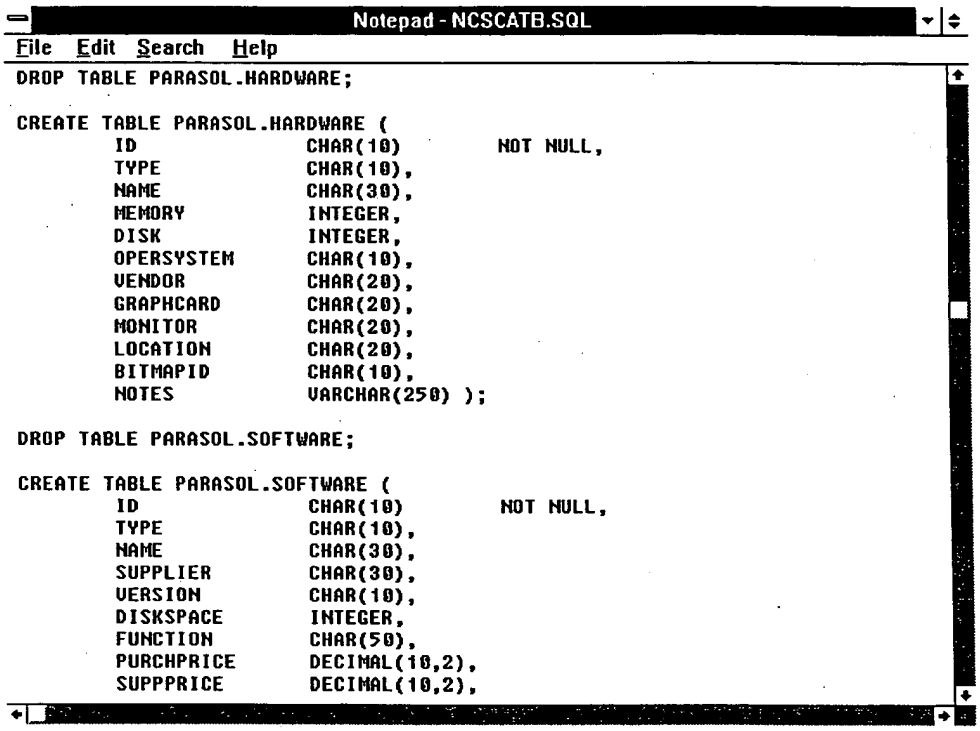

Figure 3-7: Sample SQL file for creating the Hardware and Software tables.

Parasol assumes that application tables have one unique key. Duplicate keys MUST not be allowed. In order to enforce this on inserts, the table must be given a unique key. Make sure that the key field is defined with the column definition qualifier NOT NULL.

3.3.3 Generate the Tables

For Database Manager and WatcomSql, run the XYZCREAT procedure from within those products using the interactive SQL features. Refer to the documentation for these products for further details. The first time you run the procedure, the DROP TABLE statements will not find the tables to drop for either the Parasol system tables or the user defined tables you defined in the file above. This will generate an SQL errors that you can ignore.

For example, if you are using Watcom SQL, following these steps:

- Use the Database Tools Icon to initialize a database
- Use the Interactive SQL Icon and connect to the project database using the username DBA with a password SQL (initial access to the database)
- Open the file call XYZCREATE.SQL found in the directory \XYZ\REX\
- Execute that file by selecting the Execute button
- Ignore any SQL error messages that appear about a table not existing
- Exit Interactive SQL and return to Program Manager for Windows

For XDB and SQLBase databases, run the following files in sequence to create and load your system tables.

- XYZCRTB.abc
- XYZCDTB.abc
- XYZCATB.abc
- XYZLOAD.abc
- XYZDATA.abc
- XYZHELP.abc
- XYZCRIDX.abc

Remember abc varies by database. For Gupta SQLBase it is WTS and for XDB it is SQL. Note also that for XDB, the DATA file has two parts to load, DATA1 and DATA2.

This process will create all the system and application tables for your Parasol application. If any errors are detected or generated, review them and correct the offending command file to fix the problem. Remember that the DROP TABLE statements will generate error messages when run for the first time. Refer to further details about SQL procedure statements in your documentation for the SQL database you are using.

3.4 Naming Conventions

A Parasol application is designed to be object-oriented. You will be dealing with many objects in a typical application. The naming convention used in all the remaining application development steps are described below. However, these names are automatically maintained by Parasol for you. The following technical details are provided for reference only, in the event you need to extend your application beyond default behavior. You can use another naming convention, as long as you are consistent, and remember the names you use for creating all the objects associated with your application.

Unless you need these details, you may skip to the next chapter to continue creating your application.

3.4.1 Parasol "C" Objects

A Parasol application, excluding the dialogs, table views, and SQL database access objects, consists of the following "C" objects:

- AMM - Application menu manager object.
- XYZ - Application object that inherits the Parasol object where XYZ is the name of your project. Top most object in the hierarchy of objects.
- ARM - Application record manager.
- TBF - Application tree building callbacks.

These objects will be generated by the Parasol Source Code Generator. The names are fixed.

Each table also has the following application objects:

- One or more data entry and editing dialog boxes that make all the fields of a specific table available for entering new information, or changing data records already in the database. Each dialog will be represented as one "C" object.
- A "find" dialog box object creates SQL "where clauses" and allows a user to do searches on the table. This dialog will be represented as one "C" Object.

- A Parasol Table View dialog box object that presents a list of records for a selected table when the "find" dialog box retrieves records from the table. This dialog will be represented as one "C" Object.
- A Report dialog box object allows a user to run a report defined in an external report writer. This dialog will be represented as one "C" Object.
- An Abstract Database Record "C" Object that provides the abstraction layer between the dialog boxes and the database.
- An SQL Interface Object that provides the host interface language to the SQL database that you are using.

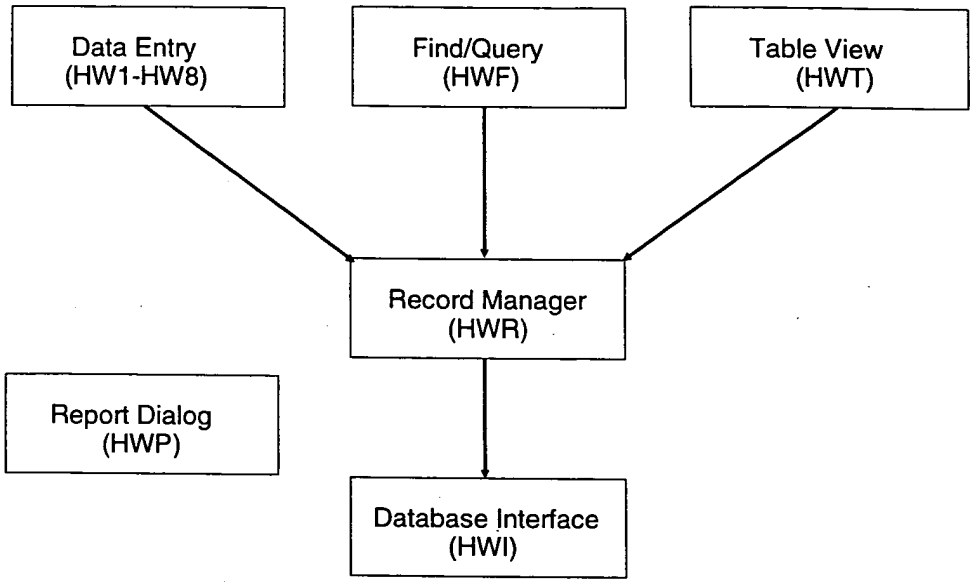

Figure 3-7: Parasol Dialog Objects with example names.

Each table will have a minimum of 6 "C" objects. You should use the following naming convention for these objects. Each object has a three character name.

The first two characters are some short-cut abbreviation of the table name, and the last character of the three character object name is:

- "1" through "8" for Main Data Dialog and satellites
- "F" for the Find Dialog
- "T" for the Table Dialog
- "P" for the Report Dialog
- "R" for the Abstract Database Record
- "I" for the SQL Interface Object

If we take the example table HARDWARE, our names would become.

- HW1 thru HW8 Data dialog objects
- HWF Find dialog object
- HWT Table view object
- HWP Run report object
- HWR Abstract Database Record object
- HWI SQL Interface object

Parasol Developer automatically names objects according to the above conventions provided one specifies Use Default Naming which is the default.

If you want to create more than one tabular report per table, further details are described in the chapter on report generation later in this manual.

3.4.2 Library Definitions File for "C" Objects

When you created the project with the Parasol Source Code Generator, it created a file named LIBXYZ.H where xyz is the name of the project. The file is in the \DSH directory of your project XYZ directory. Parasol Developer automatically maintains this file as you build and/or change your application.

This file contains a reference to every "C" object in the application. Each object that you will create using the Parasol Source Code Generator must have an entry for the following:

- ZZZ_NODEBUG - substitute ZZZ for your object name
- ZZZ_MACROS - substitute ZZZ for your object name
- ZZZ_CHECK_WORD, with a unique number for each check word - substitute ZZZ for your object name
- PZzz struct StructureDescriptionName * - substitute Zzz for your object name, and StructureDescriptionName for your object C structure name, which is automatically assigned by the Source Code Generator.

Also, each table you have defined that has a user interface must have an entry in this file. A line of the syntax shown is created for each table name in your application. The table name is the name as defined in the CREATE TABLE statement you edited in the previous chapter. This statement is automatically created by the Source Code Generator.

```
#define ITT_YOURTABLENAME   200
```

The number is unique for each table and is used to give a unique identifier to each table in a Parasol application.

3.4.3 Resource Definitions File

When you created the project with the Parasol Source Code Generator it created a file named XYZDEF.RH where xyz is the name of the project. The file is in the \DSH directory of your project XYZ directory. Parasol Developer maintains the contents of this file as you build and/or change your application.

This file contains your naming convention for dialog controls, menu items, dialog panels, bitmaps, accelerator tables, and icons. The naming conventions can be changed to suit your preferences but then you must maintain this file yourself.

This file provides a name for all your dialog panels:

- Data dialogs
- Find dialogs
- Table View dialogs
- Report dialogs

Parasol Developer will also establish names of menu items for:

- Find dialogs
- Report dialogs
- Add data dialogs
- Add child data dialogs
- Add sibling data dialogs

This file is used by Parasol Designer (Application Framework Editor) as the resource definitions file for all the objects in your application.

Chapter 4 - Setup Tables

The next phase in creating an application in Parasol Developer is to register the tables that you will use in it. This chapter explains how to register the application tables in the database.

4.1 Starting Parasol Designer

Select with a single click (not a double click) the Parasol Designer icon in the Parasol Group or Folder you created in Chapter 3.1. For OS/2 select the Object Settings for that icon from the menu options. For Windows, select the Properties option of the File menu when the icon is selected.

Ensure that the icon runs with 3 parameters and set the working directory to \XYX\DSH where xyz is the name of the project you are working with. Parasol Designer has 3 parameters and they are defined as follows:

- 1. Name of database product (DBM, GUPTA, WATCOM, XDB)
- 2. Name of database
- 3. Name of project

For a project XYZ and a database name of XYZDB and the database product XDB the parameter string on the icon would read [XDB XYZDB XYZ]. Note that when using WatcomSql or SqlBase the database name is the full path name of the database file. The other databases have a database catalog and only the database name is used.

Exit this dialog and save the new parameters.

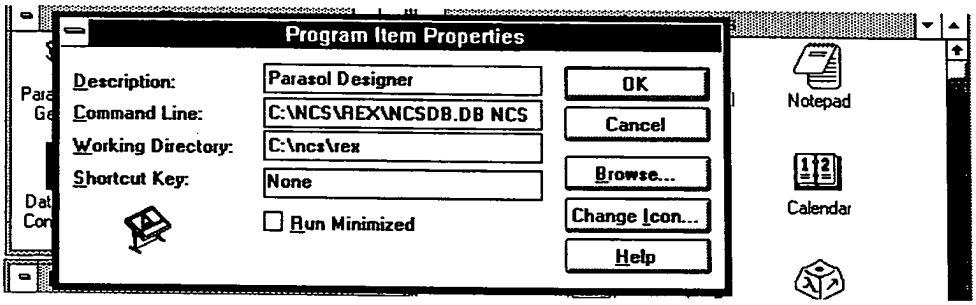

Figure 4-1: Properties dialog box for Parasol Designer

Double click on the Parasol Designer icon and you will first see the startup screen. There is only one menu to work from, the File Menu.

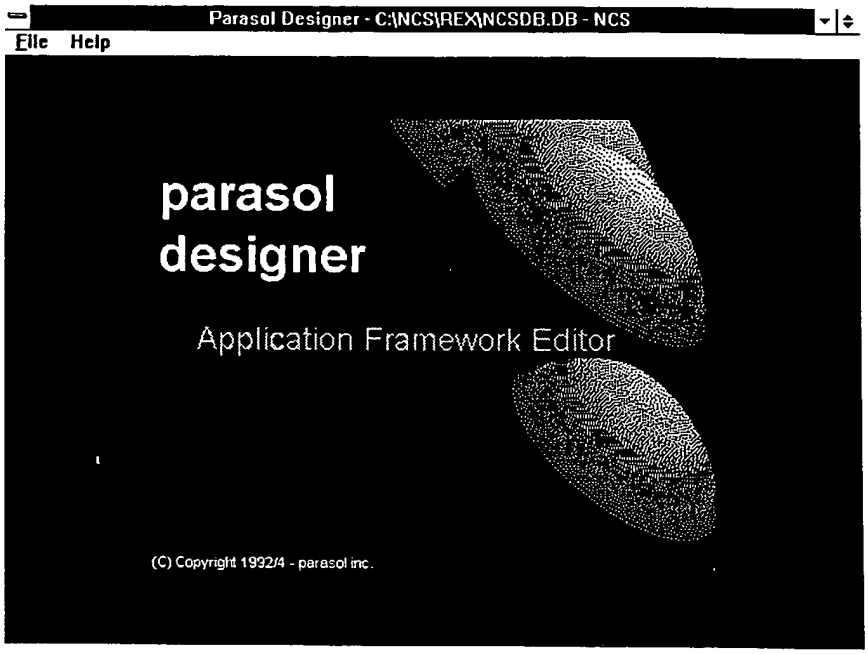

Figure 4-2: Parasol Designer startup screen.

4.2 Read in Dialog and Control Naming Definitions

Every dialog panel (data, find, table view, report) must be given a name for the Parasol Executive dialog manager is to communicate with the dialog when you run your application.

The file called XYZDEF.RH in the \DSH directory where xyz is the name of the project you are working with contains the object names for the dialog panels and dialog controls, as well as other system objects such as bitmaps, pointers and menu items.

Parasol Designer maintains this file with a name on a per table basis for 8 Data Dialog Names, 1 Find Dialog Name, 1 Table View Name, 1 Report Dialog Name, 1 Abstract Record Name, and 1 SQL Interface Name.

Therefore every time you start Parasol Designer, the only option available on the File Menu will be the Open Definitions option and the Table Specs option (other that exit).

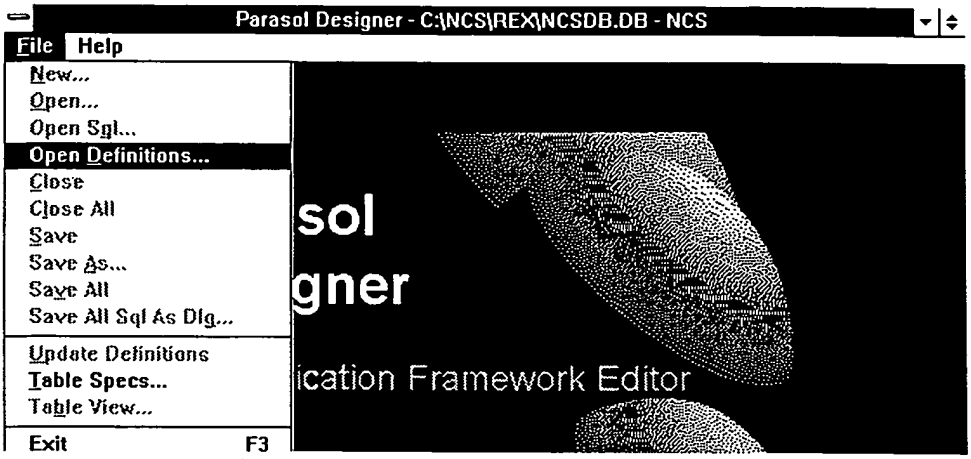

Figure 4-3: File Menu in Parasol Designer with Open Definitions highlighted.

Select this option and the Open Definition dialog box will appear for you to select the definitions file. It will have the name XYZDEF.RH in the \DSH directory of your project, and will appear in the dialog box list.

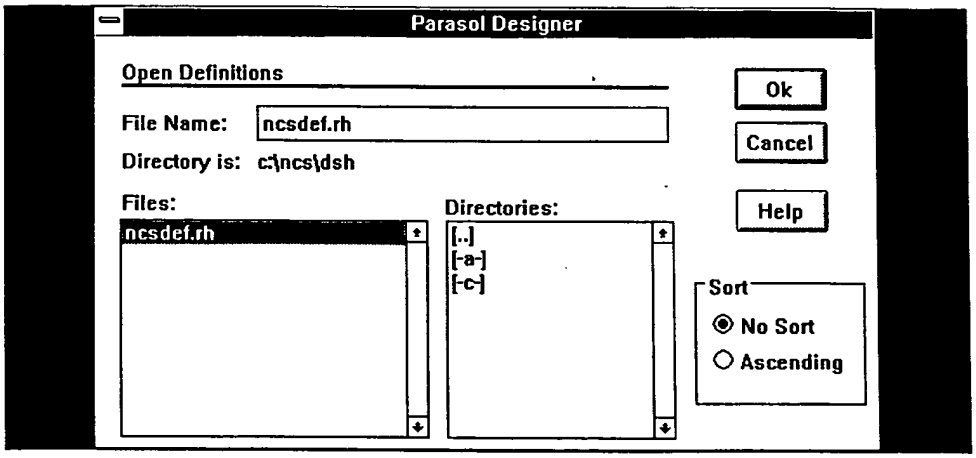

Figure 4-4: Open Definitions dialog box.

Select the definitions file and click on the OK button to use it.

4.3 Register Application Tables Type

The next step is to register the application tables in your database with Parasol. These are the tables you defined earlier in the SQL command file for your database.

To do this, select the Table Specs option of the File Menu in Parasol Designer. A dialog box for registering your data tables will appear as shown below.

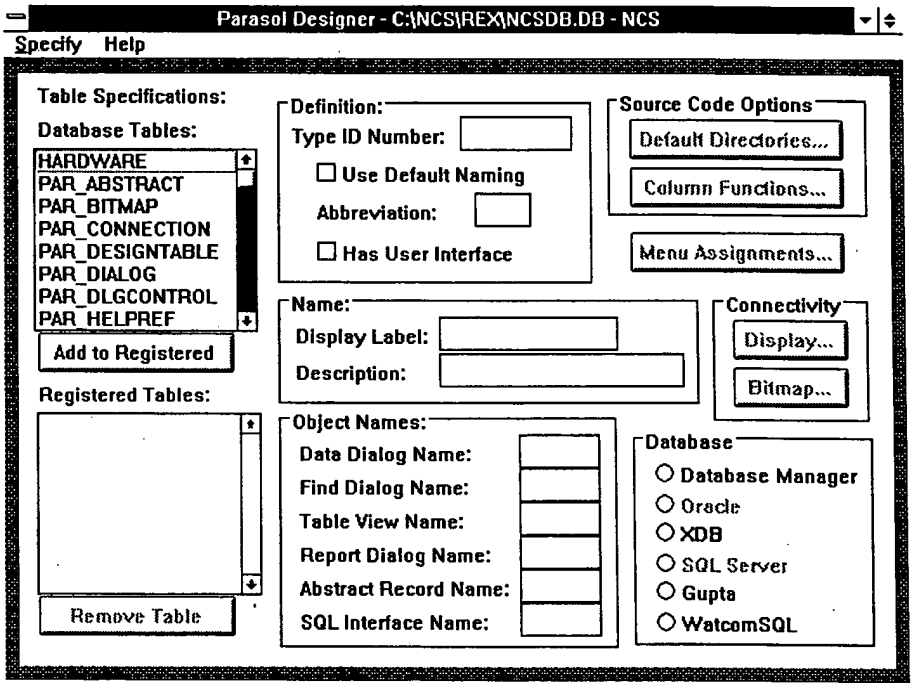

Figure 4-5: Dialog box to register details for application tables.

A list of the tables defined in your database appears in the upper left corner list box. The lower left corner list is the tables that are registered thus far in Parasol Designer. As each application table is registered, Parasol Designer assigns a unique ID number. You must then assign a two character code to that table. Then as you tab out of the abbreviation field to fill in the remaining details for each table.

Fill in the following data values for each table (sample information for the NCS application is given in square brackets [ ]):

Name:

Display Label:Name of table as it appears to a user a Parasol Application [Hardware]

Description:The description of the table. This is not displayed in Parasol. [Computers & Peripherals]

Source Code Options:

Select the Column Functions push button.

Maximum on Column:Name of column for SQL maximum function. Parasol requires a maximum function for automatic ID assignment. Select the column that represents the unique key for the table [ID]

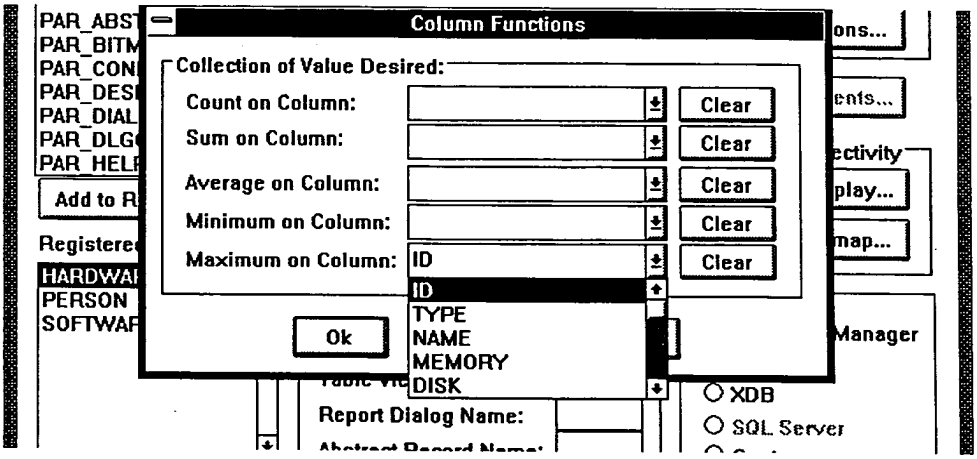

Figure 4-6: Selecting the ID column for a maximum value.

Connectivity:

Select the Display push button and supply the following:

Card Display Column:Column name that will be used by the "C" object TBF in creating the id for an item that will display in Parasol. [ID]

Description Column: Column name that will be used by the "C" object TBF in creating the description for an item that will display in the outline view and the item detail line. [NAME]

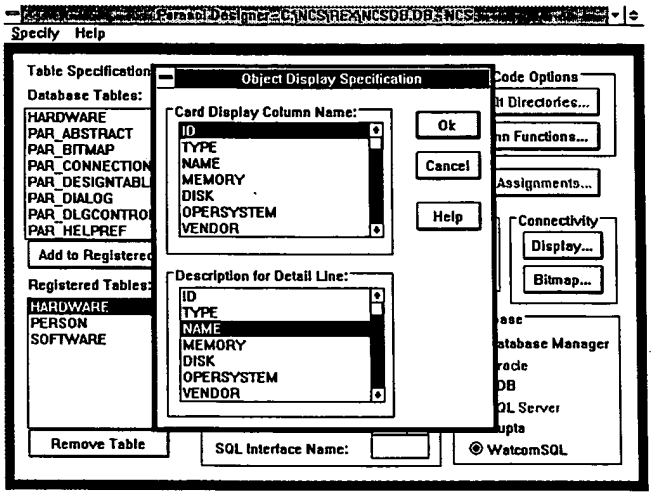

Figure 4-7: Selecting the Card Display and Detail Description columns.

Bitmap Column If Used (Optional):

Select the Bitmap push button and supply the following:

Bitmap Column: Column name that will be used by the "C" object TBF in creating the id for a bitmap to attach to a record in a table. [BITMAPID]

Bitmap Caption: The caption that appears on the bitmap display window of Parasol. [Device Picture]

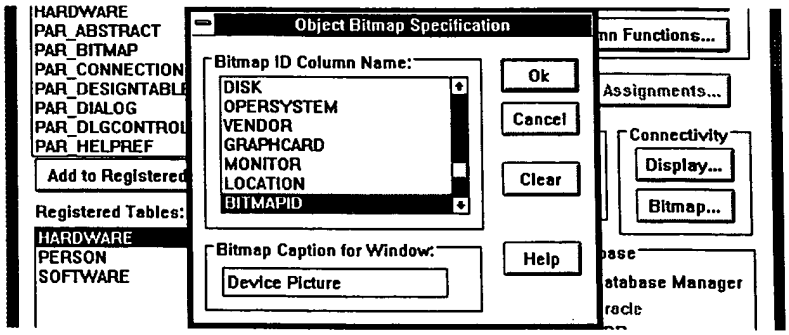

Figure 4-8: Selecting the Bitmap ID column and giving it a caption.

Database

Select the type of SQL Database Interface that you want the Source Code Generator to generate.

Select the Specify Menu and click on the Edit Dialogs option to exit the table registration function and return to the dialog design functions of Parasol Designer.

When you exit the table registration function, the Resource Definitions file XYZDEF.RH in the \DSH directory will be automatically updated with a name for each table, plus 8 Data Dialog Names, 1 Find Dialog Name, 1 Table View Name, 1 Report Dialog Name, 1 Abstract Record Name, and 1 SQL Interface Name. See sections 3.4.1 & 3.4.3. for the default names created.

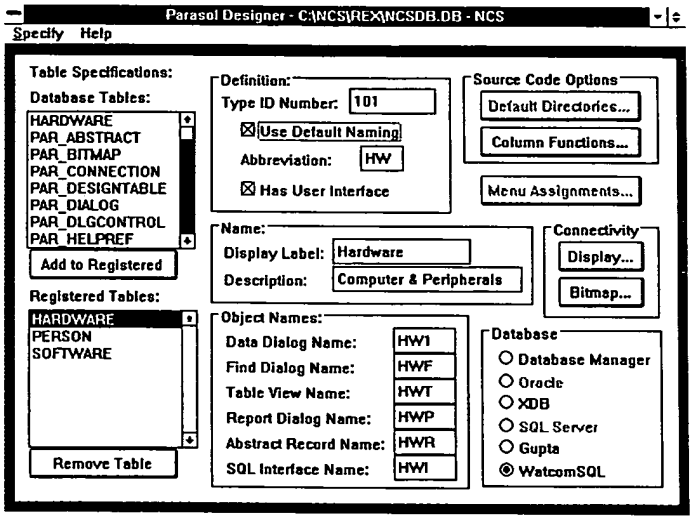

Figure 4-9: Automatic naming of the objects using default names.

4.4 Working without Default Naming

In some cases where the developer needs to assign names manually, you can enter your own names for each object. First click the check box Use Default Naming off, and then you can change the object names to your own convention. The purpose of each object is listed below, and described in further detail in Sections 3.4.1 and 3.4.3.

Object Names:

Data Dialog Name: Name of the "C" file for the main data dialog object

Find Dialog Name: Name of the "C" file for the find dialog object

Table View Name: Name of the "C" file for the table view dialog object

Report Dialog Name: Name of the "C" file for the run reports dialog object

Abstract Record Name: Name of the "C" file for the abstract database record object

SQL Interface Name: Name of the "SQC" file for the SQL database interface object

4.5 Backup Table Registration Definitions

The database table registrations just defined is stored in the database table PAR_DESIGNTABLE, which is a system table for Parasol Developer (see Appendix A). It is good practice to export this information to a disk file using the export utility of your database system. In this way, if you go back and recreate the database interactively during the application's life-cycle, you can reload the table PAR_DESIGNTABLE with your specifications.

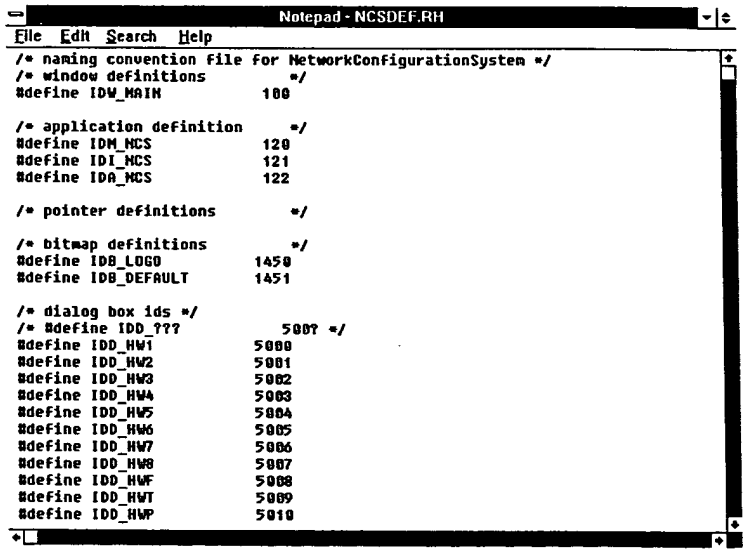

Figure 4-10: Listing of the NCSDEF.RH file after Parasol created all the object names for the three tables defined.

Chapter 5 - Specify Application

The third phase in creating an application in Parasol Developer is the major design operation for the process. In this phase, you will be creating various dialog boxes for your application, and specifying them with Parasol Designer (Application Framework Editor). This mechanism is a separate program from the Source Code Generator of Parasol Developer.

This chapter will describe how to use Parasol Designer to design the dialog boxes you need, and then how to link them to the fields and tables in your database. This process is the most time consuming part of building an application, and also the most creative.

You should also establish some standards for designing your dialog boxes so they all have a consistent "look and feel" throughout your application. For example, you may always want to have the Ok, Cancel and Help buttons placed at the bottom of a dialog box. Once you have designed these dialogs, you are ready to start building them in Parasol Designer.

There are three sub-steps to designing the dialog panels:

- Graphical design of dialog panels - placement of fields and controls on the panel
- Linking of dialog panels - linking of fields on the panel to fields in the database
- Ordering of TAB key - default sequence that the TAB key will follow on the panel

Report Dialogs are a special case and the fields are not bound to the database but to the variables of an SQL statement.

5.1 Specify Dialogs Using Parasol Designer

Now that you have identified all the objects (tables, dialogs, etc.) for your application and given them names, you are ready to start designing the details for how your application will look. This section describes how to do this using Parasol Designer, the Application Framework Editor.

5.1.1 Starting Parasol Designer

Select with a single click (not a double click) the Parasol Designer icon in the Parasol Group or Folder you created in Chapter 3.1. For OS/2 select the Object Settings for that icon from the menu options. For Windows, select the Properties option of the File menu when the icon is selected.

Ensure that the icon runs with 3 parameters and set the working directory to \XYX\DSH where xyz is the name of the project you are working with. Parasol Designer has 3 parameters and they are defined as follows:

- 1. Name of database product (DBM, GUPTA, WATCOM, XDB)
- 2. Name of database
- 3. Name of project

For a project XYZ and a database name of XYZDB and the database product XDB the parameter string on the icon would read [XDB XYZDB XYZ]. Note that when using WatcomSql and SqlBase the database name is the full path name of the database file. The other databases have a database catalog and only the database name is used.

Exit this dialog and save the new parameters.

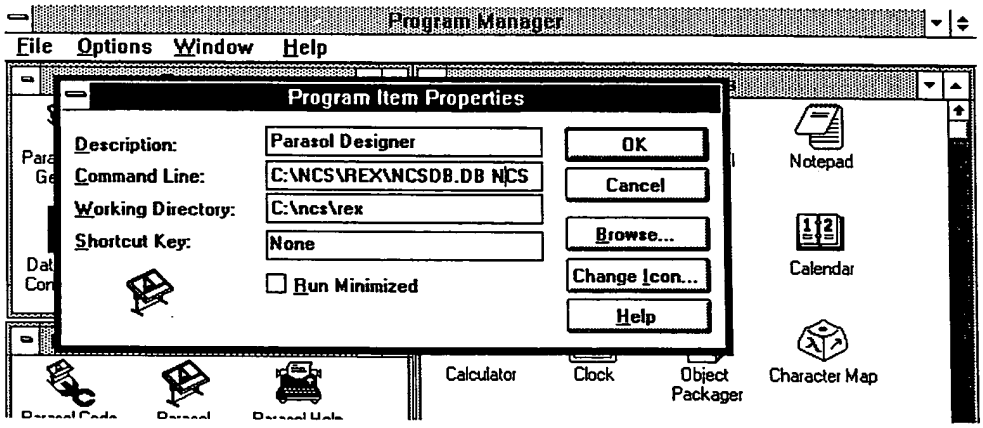

Figure 5-1: Properties dialog box for Parasol Designer

Double click on the Parasol Designer icon and you will first see the startup screen. There is only one menu to work from, the File Menu.

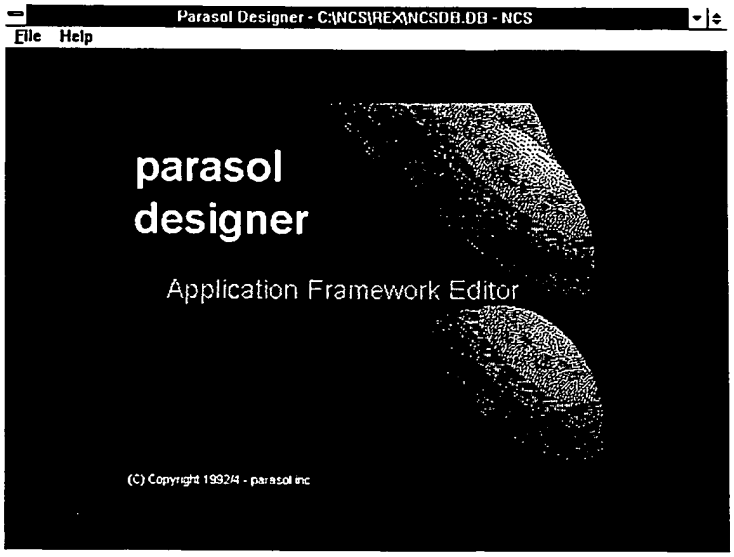

Figure 5-2: Parasol Designer startup screen.

5.1.2 Read in Dialog and Control Naming Definitions

Before attempting to create a dialog panel, your naming convention file must be created for the project. This is what you did in section 3.4.3. A dialog panel contains a number of objects. These are going to be a combination of fields (for entering data), control buttons (Ok, Cancel, etc.), and special functions (list boxes and combination boxes).

Every dialog panel and every dialog object must be given a name for the Parasol Executive dialog manager is to communicate with the dialog or dialog controls when you run your application. So the dialog panel itself, and any controls that will be displaying data or accepting data from a user, MUST be named. Parasol Designer automatically names objects that are placed on a dialog from the objects toolbar. This is not the case for the dialog panel itself but the name only has to be selected from a list.

The definition file that you already created in sections 3.4.3 & 4.3 contains ALL the resource names you will use in Parasol Designer to design and build your dialog boxes. The file is called XYZDEF.RH in the \DSH directory where xyz is the name of the project you are working with. It contains the object names for the dialog panels and dialog controls, as well as other system objects such as bitmaps, pointers and menu items.

Every time you start Parasol Designer, the only option available on the File Menu will be the Open Definitions option and the Table Specs option (other that exit).

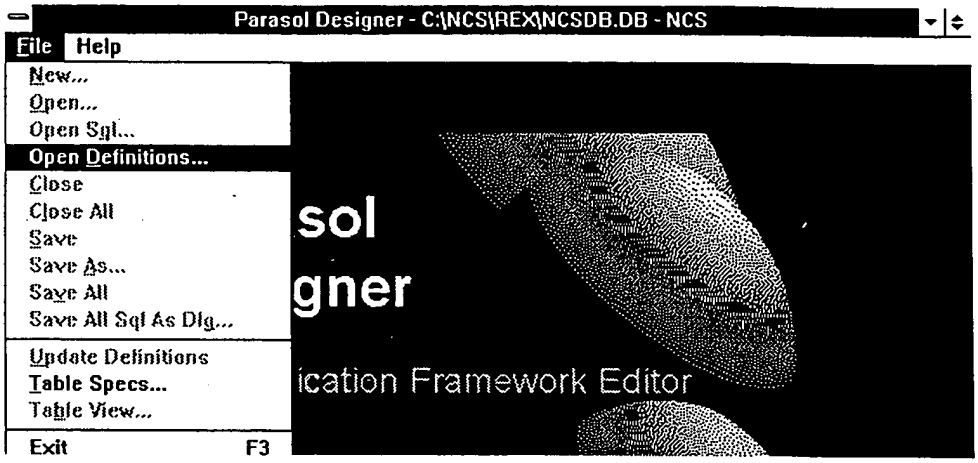

Figure 5-3: File Menu in Parasol Designer with Open Definitions highlighted.

Select this option and the Open Definition dialog box will appear for you to select the definitions file you updated. It will have the name XYZDEF.RH in the \DSH directory of your project, and will appear in the dialog box list.

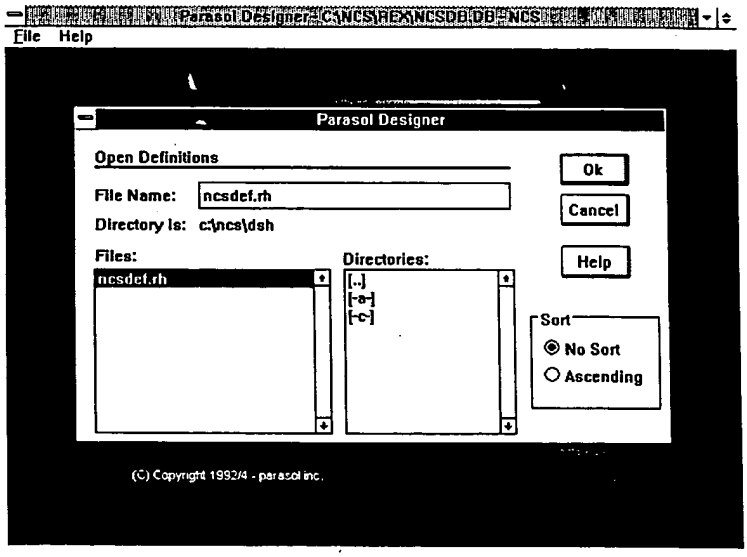

Figure 5-4: Open Definitions dialog box.

Select the name from the list you want by clicking once on it and clicking again on the Ok button, or you can double click on the name itself. That file will be loaded into Parasol Designer, and you will return to the main Parasol Designer screen with the other File Menu options now available. You will use these options to open and design the dialog boxes.

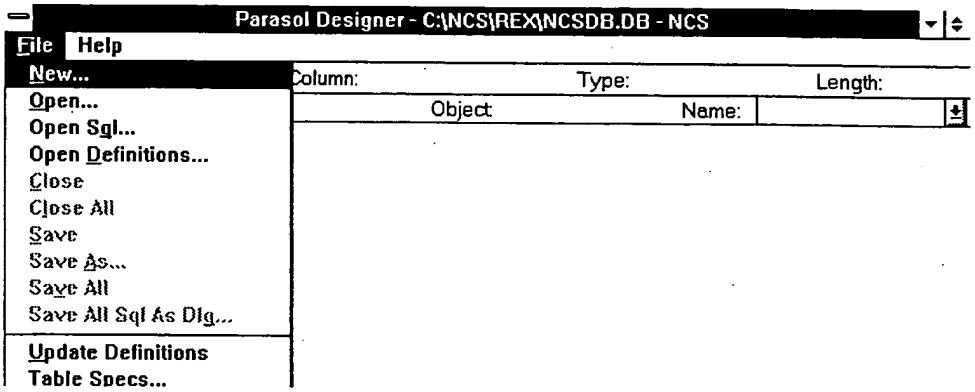

Figure 5-5: File Menu with other options now available.

5.1.3 Create Dialog Panels

With the definitions file now open you can start creating dialog boxes. Select the File Menu option New to start a new dialog panel. When you select the New option on the File Menu, the Create New Dialog screen will appear.

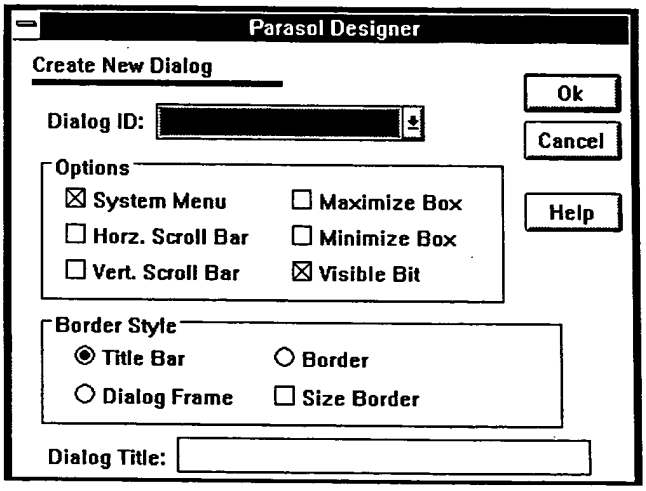

Figure 5-6: Blank dialog box for creating a new application dialog panel.

Click on the downward pointing arrow for the Dialog ID box to see a list of the names you have defined. Remember, this list contains all the objects defined. You want to select the one for this dialog panel. It will start with the code IDD (for ID Dialog), have a two-character code for the table, and a single character code representing the type of dialog box (1-8, F,P, or T). Select the name you assigned in your resource definitions file for the panel you want to design at this time.

The OS/2 and Windows application quidelines recommend that a dialog box have a system menu, a title bar and a dialog border for a frame (the defaults). You can also select other options by clicking in the box adjacent to each name. The options are:

OS/2	Windows	Description of Option
Title Bar	Title Bar	the dialog box will have a title bar at the top
System Menu	System Menu	a File menu will be created in the dialog box
Horz Scroll Bar	Horz Scroll Bar	the dialog box will have a horizontal scroll bar
Vert Scroll Bar	Vert Scroll Bar	the dialog box will have a vertical scroll bar
Visible Bit	Visible Bit	this switch makes the panel visible (required)
No Byte Align		allows sizing accurate to one pixel
Sync Paint		{not currently used}
Min Button	Minimize Box	the dialog panel will have a minimize button ava
Max Button	Maximize Box	the dialog panel will have a maximize button ave
Size Border	Size Border	the user will be able to resize the borders
Border	Border	the dialog box will have a border
Dialog Frame	Dialog Frame	the dialog box will have a frame around it

If you click on the Title Bar option, the Dialog Title name box will become available to you. Enter the name for this dialog box that you want to appear at the top of the box when the application runs.

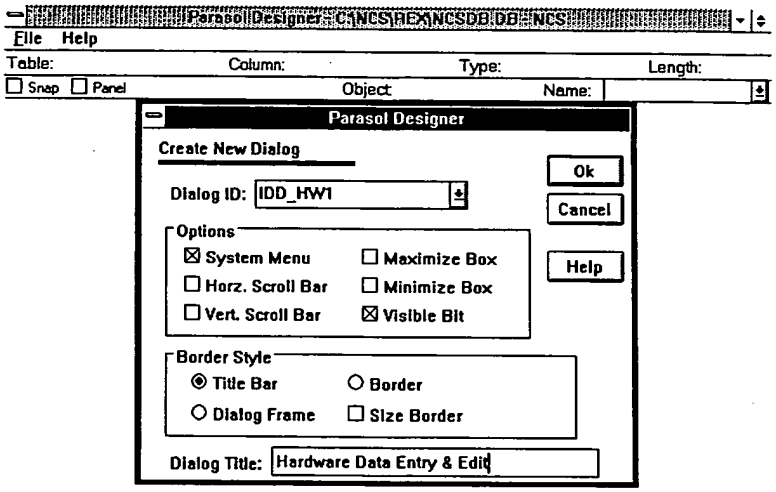

Figure 5-7: Completed Create New Dialog options for the Hardware table.

After you have completed all the information, click on Ok and you will return to the main Parasol Designer screen. A blank default box will appear in the middle of the screen, and the Parasol Designer toolbar icons will appear down the left side. This is where you will design the details for this dialog box.

You will create dialog panels for each table for your:

- Data Dialogs
- Find Dialogs
- Table View Dialogs
- Report Dialogs

After you have created a blank dialog panel, you can store it as either a file on disk or as a record in the SQL database. Then later, you can use the Open option to open an existing dialog specification file from disk, or use the Open Sql option to open an existing dialog specification that you have stored in the SQL database. You MUST store dialogs in the SQL database in order to make the links between the objects in the dialog panels, and the corresponding fields and tables in the database.

5.1.4 Place Dialog Controls on Panels

When you create a new dialog box, the main screen of Parasol Designer will appear with a blank, default box in the middle and the toolbar icons down the left side which represent the various items and controls you can place on the dialog box

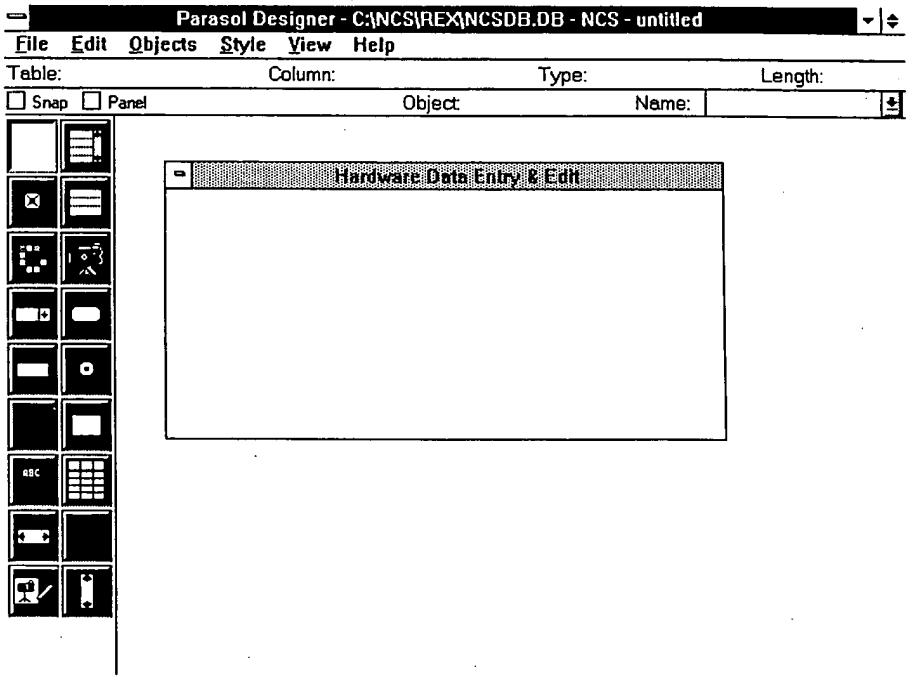

Figure 5-8: Blank dialog box with Parasol Designer toolbar icons appearing.

When you open an existing dialog box (either from disk or from the database), the current specification of that dialog box will appear.

There are two ways of placing objects on the dialog:

- Using the Objects toolbar
- Dragging objects from the Table View window

If objects are dragged from the Table View window onto the dialog the linking to the database happens at the time the object is placed. See section 5.2.2 for detail on this method of creating dialogs.

You now select a icon from the left by clicking on it, and then placing that object on the dialog box where you want it to appear. To place the item on the dialog panel, after you select it from the icon list, click on the dialog panel and drag the mouse to make the size for the object you have selected. Experiment with a few of the icons to see how they behave. Further details for each are listed below.

This process is very much like painting a picture. You simply layout the items for the dialog box much like you sketch it with a piece of paper.

The buttons OK, Cancel and Help should be included on all Data, Find and Report dialogs, but must not be used for Table View dialogs. Remember to be consistent in your designs, and follow the Microsoft or IBM quidelines as much as possible to adhere to industry standards.

The following list is in the sequence that the icons appear, from left to right, top to bottom. The name appears in the information line of Parasol Designer when you click on an icon. Parasol Designer provides the following dialog panel objects and controls to design your dialogs:

Tool Name	Description
• Selector:	Arrow pointer for selecting objects once they exist
• List Box:	Special box that contains a list of values to choose from
• Bitmap:	Location of a graphic bitmap on the dialog panel (OS/2 only)
• Multi Line Edit:	Large box where the user can enter multiple lines of text
• Check Box:	Small box with text annotation that a user can check
• Picture:	Location of a photographic image on the dialog panel
• Color Palette:	Select different colors for parts of the dialog panel
• Push Button:	Buttons with text imbedded that perform operations
• Combo Box:	Combination of data entry and/or select from a list of values
• Radio Button:	Special buttons where only one is valid from a list
• Entry Field:	Standard box for entering data for a field
• Rectangle:	Draw a rectangle on the dialog panel to highlight objects

- Frame: Draw a frame on the dialog panel to highlight certain objects
- Spin Button: Special button that "spins" through valid options (OS/2 only)
- Group Box: Combine a group of objects on the dialog panel together
- Table List Box: Special box that lists multiple records from a data table
- Horz. Scroll Bar: Place a horizontal scroll bar on the dialog panel
- Static Text: Annotation text placed anywhere on the dialog panel
- Icon: Place a special graphic icon on the panel
- Vert. Scroll Bar: Place a vertical scroll bar on the dialog panel

These options are also listed under the Objects menu of Parasol Designer, and can be selected from that menu as well as the icons down the left side of the screen.

Although this list is comprehensive, the following controls are considered part of the standard behavior of a Parasol Executive application, and are therefore supported by the Parasol Source Code Generator. Other controls not listed below will require additional C coding as discussed in Appendix C.

The currently supported controls are:

Name	Operation
Combo Box:	Field get and sets. Set text limits. Setting of color. Automatic choice list with double click access to underlying record for each choice. If a required field, a message of your choice for a missing entry. Default selection.
Edit:	Field get and sets. Set text limits. Setting of color. If a required field, a message of your choice for a missing entry.
Frame:	No behavior, a static object.
Group Box:	No behavior, a static object.
List Box:	Field get and sets. Set text limits. Setting of color. If a required field, a message of your choice for a missing entry.
Multiple Line:	Automatic get and sets. Set text limit. No validation.

Push Button: Ok (named DID_OK), Cancel (named DID_CANCEL), and Help (named DID_HELP). Push buttons can be made to run satellite data dialogs.

Rectangle: No behavior, a static object.

Table Listbox: This can only be used for the definition of table view objects in a Parasol application. For a dialog to become a table view, it must contain only one control.

Text: Field get and sets. Static field, no data entry.

The standard options that you will probably use the most are text annotations, entry boxes, multi line entries, list boxes, combo boxes, and push buttons. Push buttons are used for the Ok, Cancel, Help and to access satellite data entry screens for complex tables.

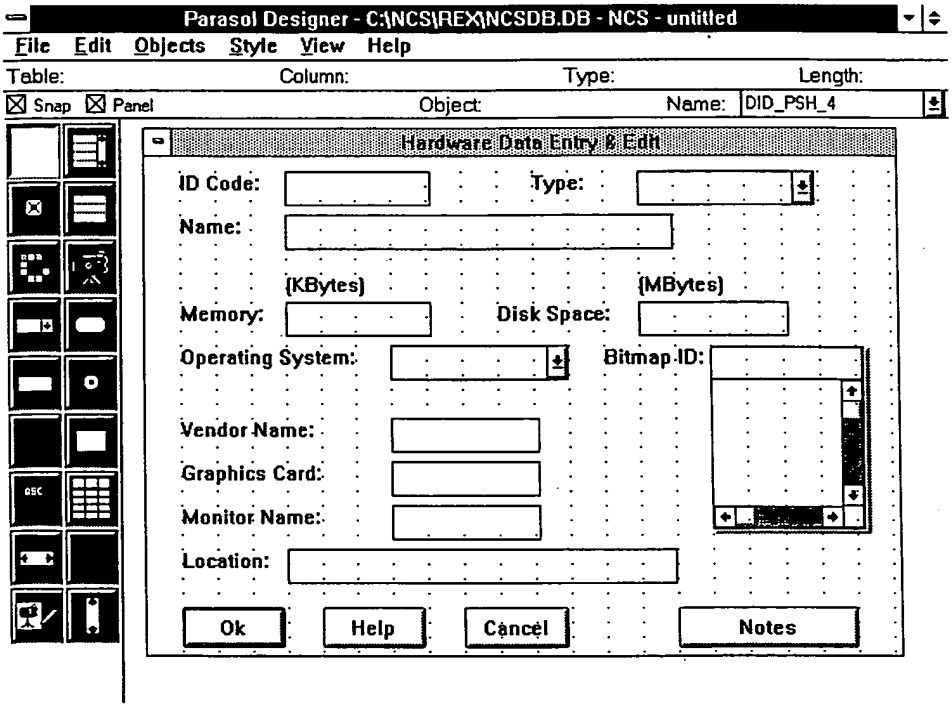

Figure 5-9: Data entry screen with several objects defined in Parasol Designer.

Please note that Table View Dialogs are a special case of a dialog panel because Parasol puts a menu on it and automatically sizes the table list control within it. A Table View panel is defined as a dialog panel with only one control in it -- a Table List control.

If you use the above controls exclusively on dialog panels, the "C" object created by the Parasol Source Code Generator will require no changes other than for non-default behavior.

Almost any behavior is possible by using the GDB (Generic Dialog Box Object) and the WND (Basic Window Object). However, these require special modifications to the source code once it is generated, and before it is compiled. See Appendix C for details of using the Parasol Programming Interface if you want to use other objects in your panels.

5.1.5 Name Dialog Controls on Panels

Notice that as objects are placed on the panel a unique name within this control panel is automatically assigned. The panel itself has already been given the object name from the resource definition file when you created it.

The menu option Edit - Reassign Names will reassign unique names within a dialog panel in the sequence that they are ordered.

When you select an object on the panel with a click of the mouse the name appears at the top right of the screen. This list is from the XYZDEF.RH file.

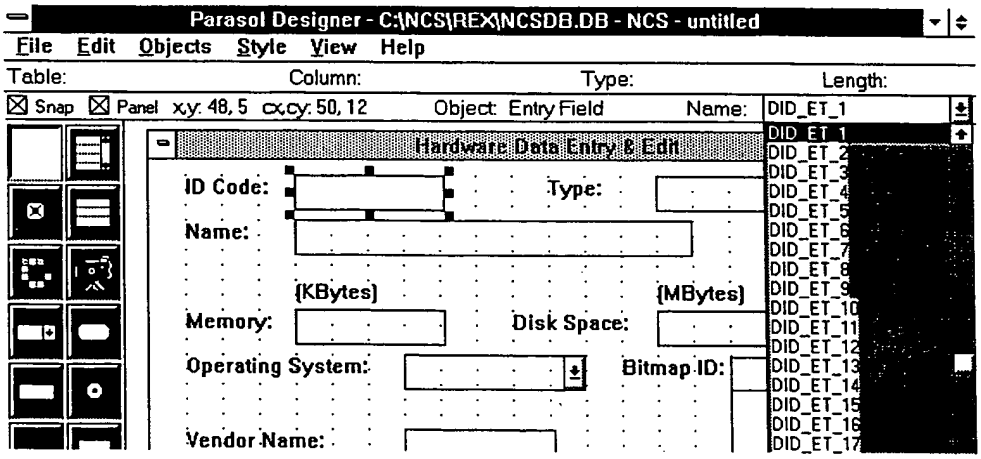

Figure 5-10: Selected data entry box with the list of names.

The names you will find in the list use the following Parasol Naming System:

- DID_ST_n - Static Controls
- DID_ET_n - Entry Field Controls
- DID_MLE_ n - Multiple Line Entry Controls
- DID_LBX_n - List Box Controls
- DID_PSH_n - Push Button Controls
- DID_COMBO_n - Combo Box Controls
- DID_CBX_n - Check Box Controls
- DID_RB_ n - Radio Button Controls
- DID_PCF_n - Picture Frame Controls
- DID_PAL_ n - Color Palette Controls
- DID_SPIN_n - Spin button Controls

For each of these, several names have already been created in the definition file when you first created your project. They are simply numbered 1-n, where n varies depending on how many of that type you would normally expect to use on a single dialog box. Note that you CAN use the same name for objects on different dialog boxes. In other words, you can use the name DID_ET_1 for the first data entry box on all your data entry dialog boxes.

For each Data Entry, Find dialog box and Report dialog box you must use the following names for these push button objects:

- DID_OK - Ok Button Control
- DID_CANCEL - Cancel Button Control
- DID_HELP - Help Button Control

These have pre-defined behavior and are used by Parasol Executive to perform those functions. The Edit - Reassign Names option will automatically give the above names to push buttons that have the text Ok, Cancel, and Help.

5.1.6 Assign Properties to Controls on Panels

You assign operation properties to each object on the panel, as needed, by first selecting the object, and then when the six black "handles" appear indicating you have it selected, you double click with the left button of the mouse or press the space bar. A Control Styles dialog panel will appear that is different for each different control type.

For each type of object, you will be able to define various properties. Some of these include:

- Text Entry: Text Alignment, Options, Default Text, ID Symbol
- Combo Boxes: Type, Text, ID Symbol (if you select the Drop Down List option, then only valid values will be permitted in this field)
- List Boxes: Types, ID Symbol
- Buttons: Types, Options, Button Text, ID Symbol (you should declare the Ok button as the default for the dialog panel to cause the operation to be performed when a user presses the Enter key)
- Static Objects: Basic Styles, Text Styles, Text, ID Symbol

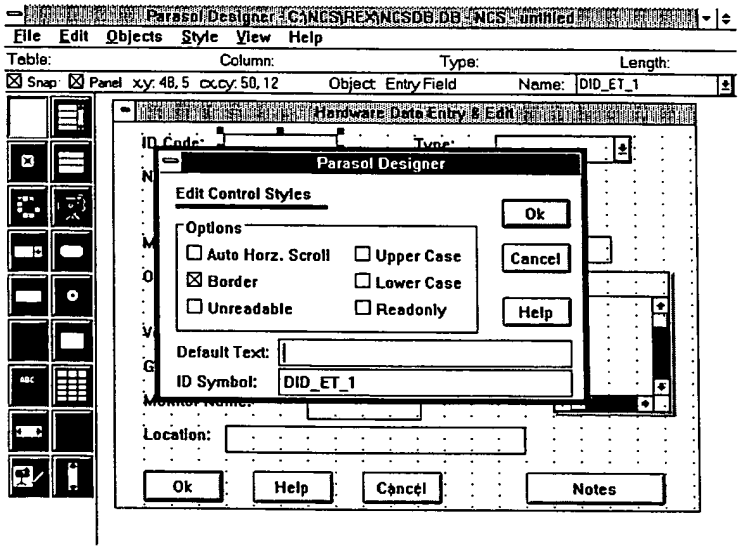

Figure 5-11: Edit Control Styles dialog box.

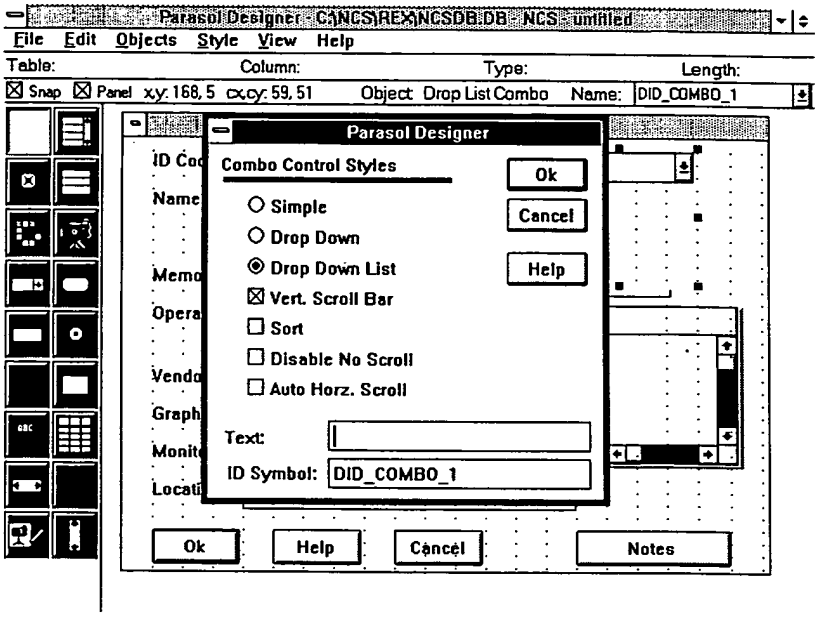

Figure 5-12: Combo Control Styles dialog box.

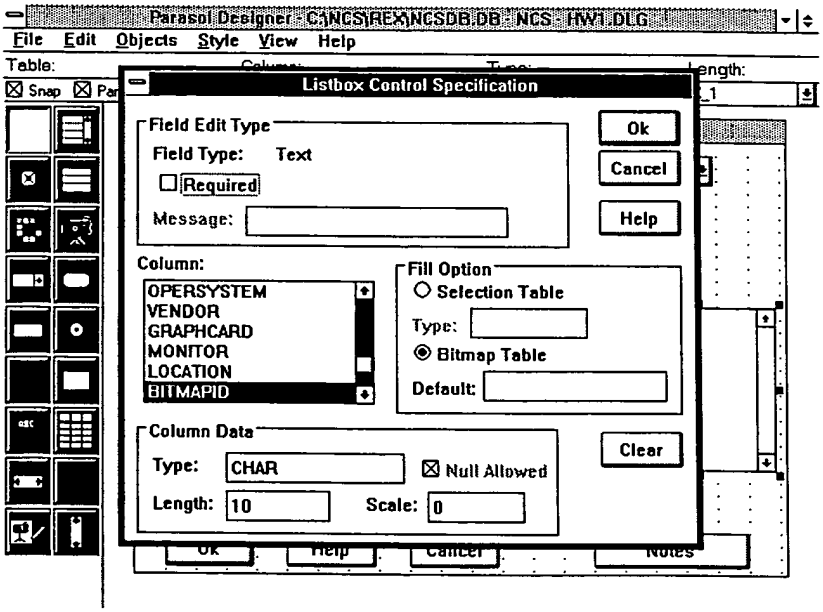

Figure 5-13: List Box Control Styles dialog box.

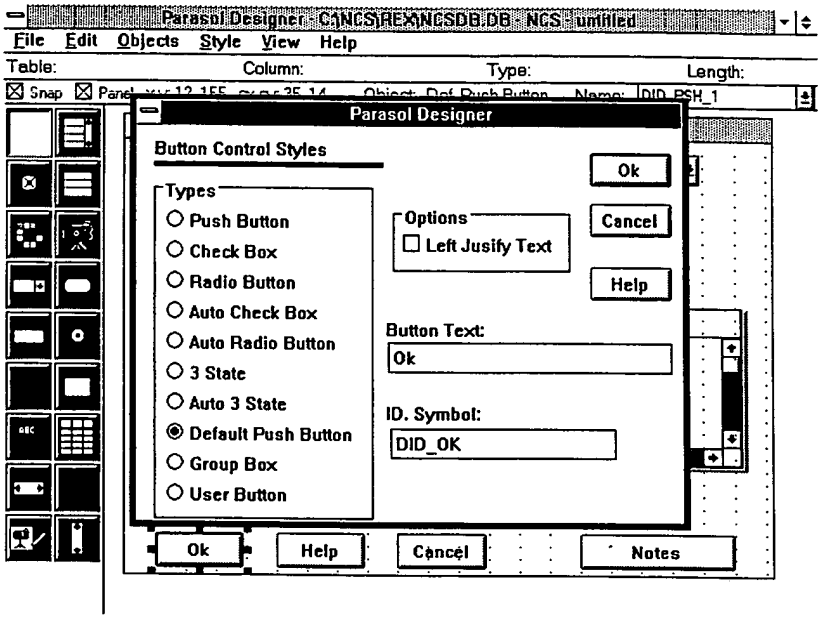

Figure 5-14: Button Control Style dialog box.

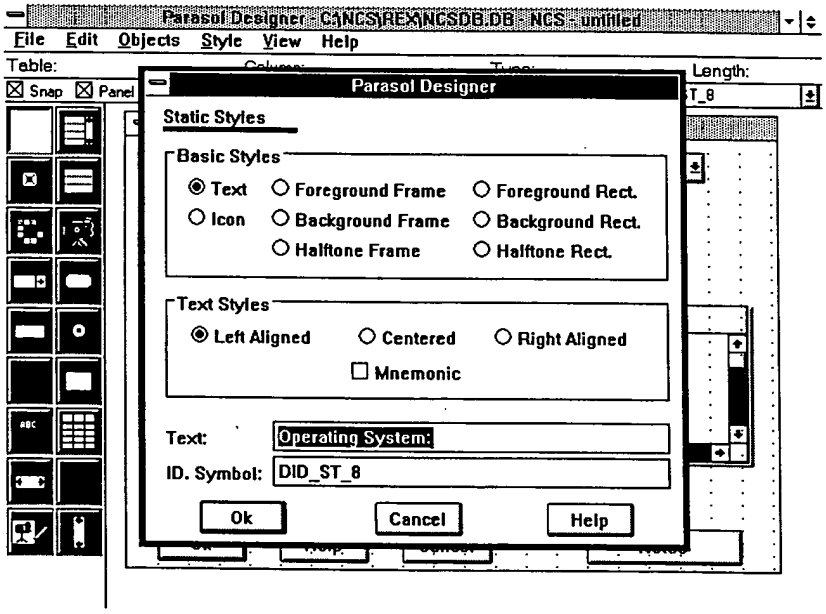

Figure 5-15: Static Control Style dialog box.

After you have selected the options you want or entered text as appropriate, select the Ok button on these dialog boxes for those properties to be assigned to that object in the panel.

5.1.7 Save Dialog Panels to Disk

Now that you have designed your dialog box, named all the objects, and given those objects operational properties, you are ready to save them. You can save them either to disk or to the SQL database.

In order to link the dialog objects to the database the dialog MUST be stored in the database. We recommend that you convert a dialog to SQL format immediately.

When you created your project, the Parasol Source Code Generator created a directory named \DSR in your project directory on your disk. This is the directory that you should use to save your dialog panels to disk.

The dialogs should always be named the same on disk as your "C" object that uses the dialog. That is the three character naming convention discussed in section 3.4.1.

Dialog panels saved to a disk file contain the naming convention and graphical definitions. You can view your dialog definition in these files with a text editor, and you can regard these as source code files.

With a specific dialog box visible on the screen, select one the Save options from the File menu.

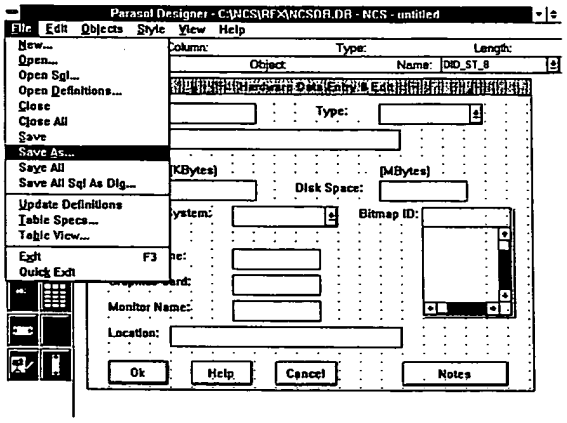

Figure 5-16: Save As option highlighted of the File Menu in Parasol Designer.

If you have just created a new dialog, use the Save As option to save it as a new name. If you opened the dialog box from a disk file, then you can use the Save option to re-write the information to disk. If you opened the dialog from the SQL database, a save will store the information back in the database.

Depending on where you opened the dialog box from, the Save As screen has a button that changes from "Convert to DLG" to "Convert to SQL". Use these buttons to change where you want to save the dialog information. Click on the Convert to DLG (if shown) to make the dialog a disk file. Click on Convert to SQL (if shown) to make the dialog an SQL record.

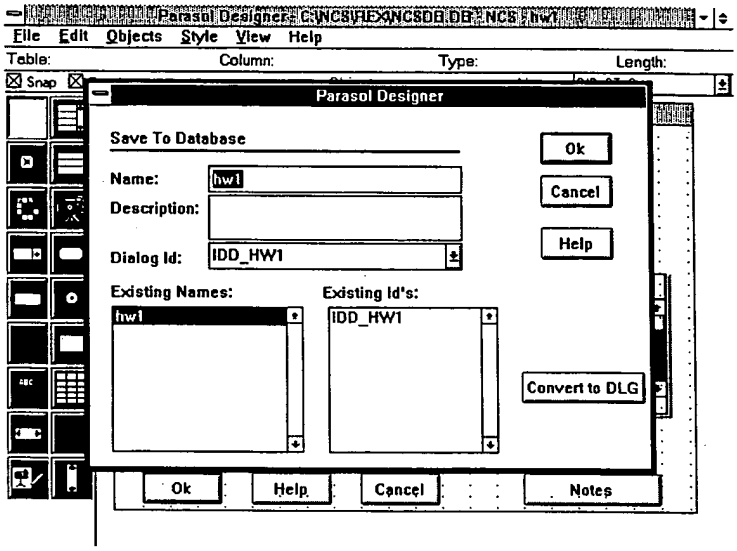

Figure 5-17: Save to Database dialog box with Convert to DLG shown.

5.1.8 Edit Order and Location of TAB and GROUP stops

The sequence of fields that the TAB key will move through when the application is running will be based on the sequence in which you created the objects on the dialog panel. The ordering of objects also has an effect on the display of objects that are on top of others. For example the Group control must have a lower sequence number than the objects inside of it (otherwise they disappear). The ordering of the objects can be edited using the Edit - Order menu.

Select the Edit - Order menu and the dialog box will change to show the sequence of the objects in the dialog panel you have just designed.

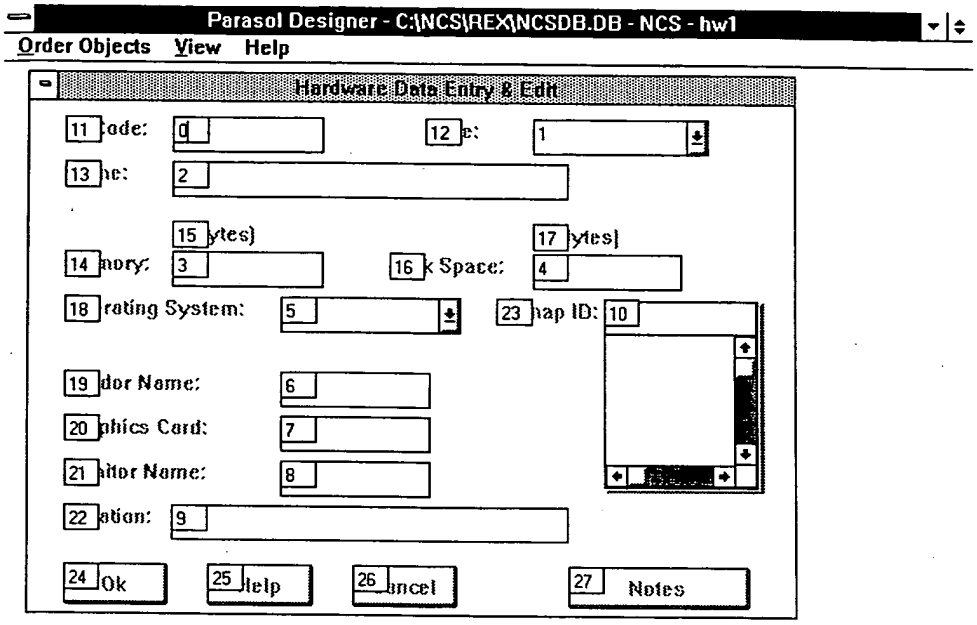

Figure 5-18: View of the sequence numbers of objects in a dialog panel.

and depending on the selection of the Specify menu the dialog will show:

- Sequence - The sequence the objects are in is placed on the objects
- Groups - Objects with group stop are checked
- Tabs - Objects with tab stops are checked

The tab and shift-tab keys move through the TAB stops and the up and down arrow keys move through the GROUP stops.

Once you have the objects in the order you want, and the tabs set, save the dialog away to the SQL database.

5.1.9 Change Dialogs

To make changes to any dialog box, simply re-open the dialog box from either disk or the database by using the Open or Open SQL options of the file menu of Parasol Designer. Remember if you have just started Parasol Designer, you must first open the definitions file, and then open the dialog box designs.

Use the select arrow icon to point at an object on the screen. Then you can:

- Delete it with the Cut option of the Edit menu
- Change its size by dragging one of the small black boxes that you see
- Send the object behind others using the Send to Back option of the Edit menu
- Bring the object in front of others using the Move to Front option of the Edit menu
- Double click with the left button to change any control style options
- Click on the list of names and change the name of the object
- Add more objects on the dialog by selecting another object icon and placing it on the dialog panel

To change information about the panel itself, rather than objects on the panel, you must first turn on the Panel check box at the upper left corner of the Parasol Designer screen. This allows you to select the panel rather than the objects on it, and make changes to the panel such as its title. Double click with the left button and push spacebar to have access to the panel style dialog box to change panel properties.

5.2 Link Dialog Panels to Table and Table Columns

If you have only saved your dialog panels to disk so far, then read in your dialog panel from the \DSR directory were you saved them in the previous section. Select the menu command SAVE AS and save the dialog panel to the database by changing the lower right button to Convert to SQL.

This will store the design specification of the dialog panels as records in the SQL database for your project. This is necessary in order to link the objects of the dialog panels to the field and table objects defined in your database for this application.

5.2.1 Assign Project Definitions to Dialog Panels

Now that the dialog panel is a database record, you are ready to attach the panel to your project database. First make sure the Panel checkbox is X'ed and select the panel. Double click the right mouse button or push ctrl-spacebar to enter details for that dialog box. You will now see the Dialog Specification dialog box if it is a Data or Find dialog box, or the Table View Specification dialog box if it is a Table View dialog box.

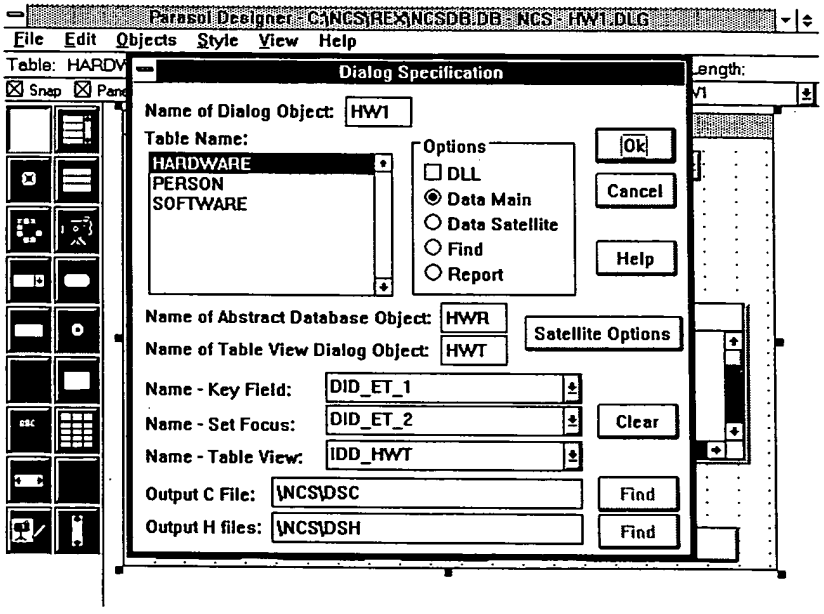

Figure 5-19: Dialog Specification dialog screen for Data Entry screen.

Select a table name that this panel is connected to. This will fill in all the fields except the parameters as described below:

- Type of Dialog:

  Data Main - Main data dialog panel

  Data Satellite - Satellite to a main data dialog panel

  Find - Find dialog

  Report - Report dialog

- Name - Key Field - Name of key field control on the panel.
- Name - Set Focus - When a data dialog is activated in a Parasol Executive application for editing the data, the key field is disabled and cannot be changed. The cursor needs to be repostioned from the key field (if its the first tab stop). This is the object that should get the focus on data change.
- DLL - Check if the source code is to be put into a Dynamic Link Library. This is an advanced user option.

If default naming has been turned off the following parameters must be filled in.

- Name of Dialog Object - Name of the object as specified in sections 3.4.1 & 3.4.3
- Name of Abstract Database Object - Name of object as specified in sections 3.4.1 & 3.4.3
- Name of Table View Dialog Object - Name of object as specified in sections 3.4.1 & 3.4.3
- Name - Table Dialog - This is the name of the Table View panel as defined in sections 3.4.1 & 3.4.3. If this table value is incorrect, the drag and drop command to copy data for table views to data dialogs will not function.
- Output C File - Directory where "C" object is to be placed.
- Output H File - Directory where "H" headers are to be placed.

For Find dialog boxes, you design the layout of the objects in the same manner. The data entry boxes represent values that will be used to build an SQL statement to search the database and "find" data records that satisfy those values. A typical find dialog panel is shown below.

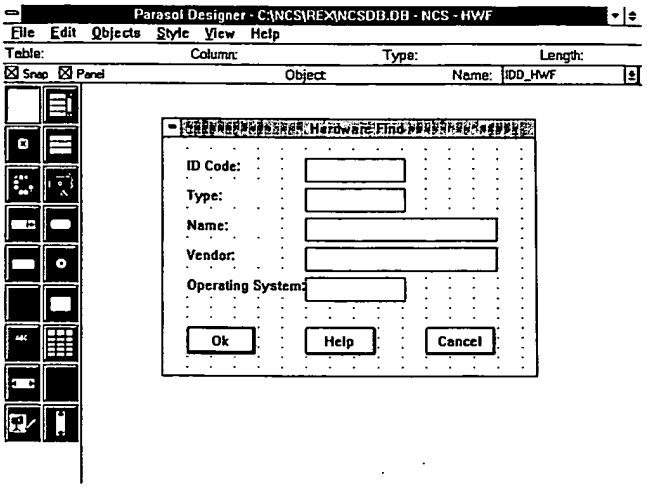

Figure 5-20: Typical Find Dialog panel design.

You identify the same parameters for a Find dialog panel as with a Data dialog. You access these parameters in the same way as before; select the panel and double click with the right mouse button.

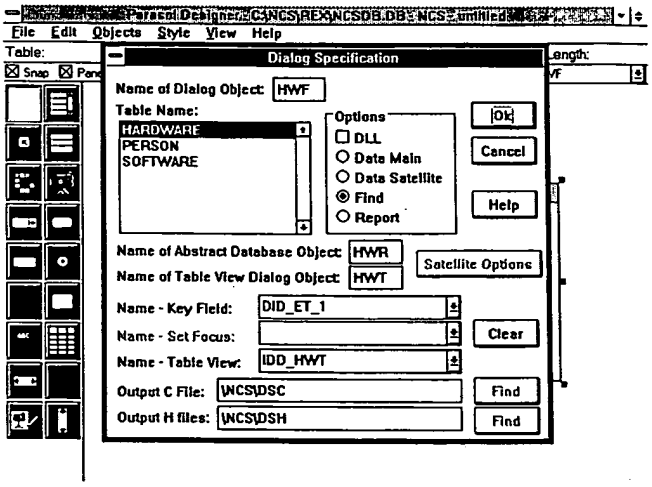

Figure 5-21: Dialog Specification dialog screen for Find screen.

For Table Views, you ONLY use the Table List Box tool to create a single object on the dialog panel. This is used to display multiple records found during a search of the database. The Ok, Cancel and Help push buttons SHOULD NOT be placed on this dialog type. A typical Table View dialog panel is shown below.

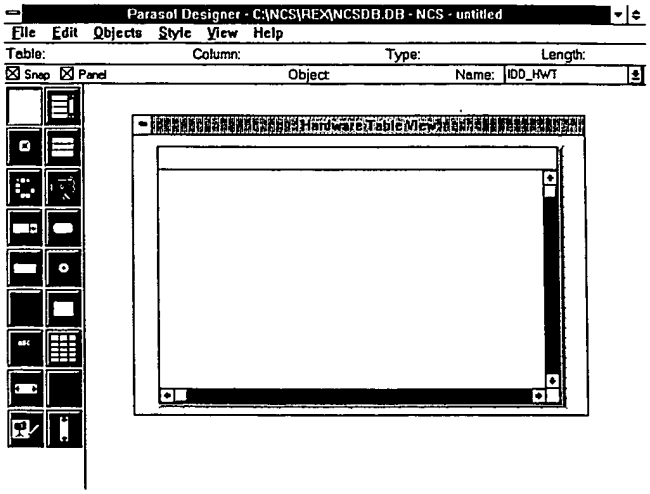

Figure 5-22: A typical Table View dialog panel.

The parameters required for this dialog type are much fewer and slightly different. You access these parameters in the same way as before; select the panel and double click with the right mouse button.

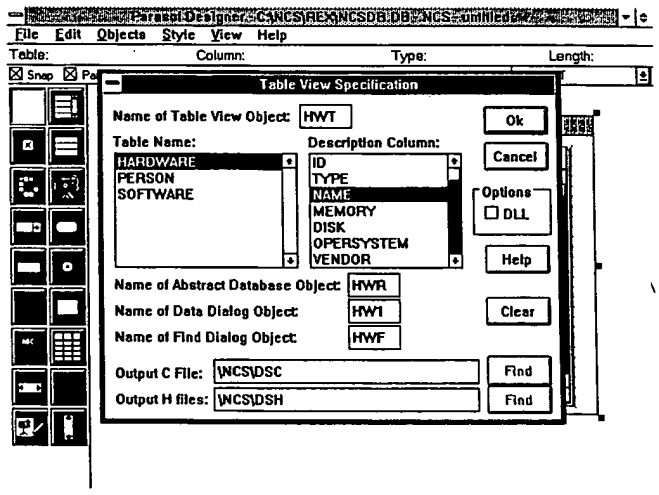

Figure 5-23: Table View Specification dialog screen for Table View screen.

Select a table name that this panel is connected to. This will fill in all the fields except the parameters as described below:

- Description Column - Column that is used to display description of an item in Parasol.
- DLL - Check if the source code is to be put into a Dynamic Link Library. This is an advanced user option.

If default naming has been turned off the following parameters must be filled in.

- Name of Table View Object - Name of the object as specified in sections 3.4.1 & 3.4.3
- Name of Abstract Database Object - Name of object as specified in sections 3.4.1 & 3.4.3
- Name of Data Dialog Object - Name of object as specified in sections 3.4.1 & 3.4.3
- Name of Find Dialog Object - Name of object as specified in sections 3.4.1 & 3.4.3
- Output C File - Directory where "C" object is to be placed. Recommend using the \DSC directory of your project name.
- Output H files - Directory where "C" object headers are to be placed. Recommend using the \DSH directory of your project name.

When you have entered all the appropriate information in these specification dialogs for each panel, select the Ok button to save. This will link the panel you are designing to the various objects in the database that it works with when the application runs.

If a data dialog panel has a satellite data dialog (for large, complex data tables where all the fields will not fit on one data dialog), you simply define the satellite data dialog similar to the main one. Then once it is saved in the database with the appropriate name (using numbers 2-8 for the object name), you then connect it to the main data dialog box using the Satellite Button on the specification dialog for the main dialog panel.

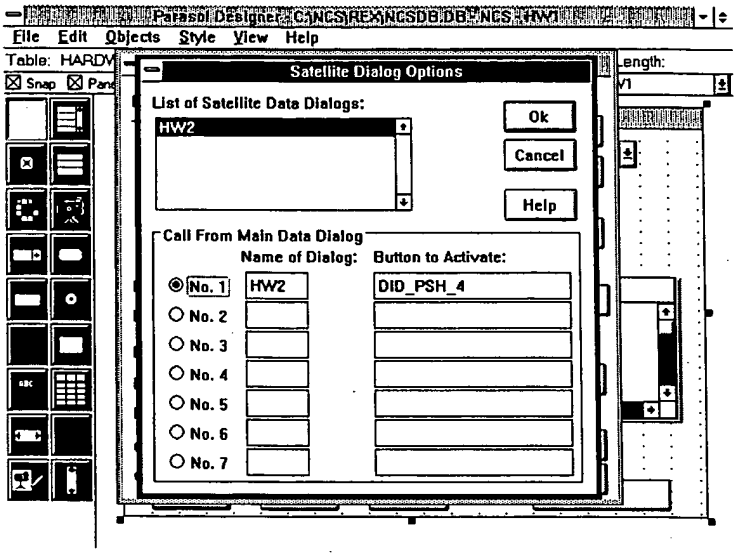

Figure 5-24: Specification of a satellite dialog panel connected to a main dialog.

For Report dialogs, the specification of the details is different from the other dialog panel types. A typical Report Dialog panel will have several data entry fields for selecting records for the report (similar to the Find Data Dialog panel), but should also have a List Box for identifying the order in which the data records are to appear in the report.

You can also place Push Buttons on a report dialog panel, usually for Run, Cancel and Help.

These values are passed to your report writer (Query Manager for OS/2 and ReportSmith for Windows) to generate the appropriate SQL statement to make the tabular report you have previously defined in those report writers. A typical report dialog panel design is shown below.

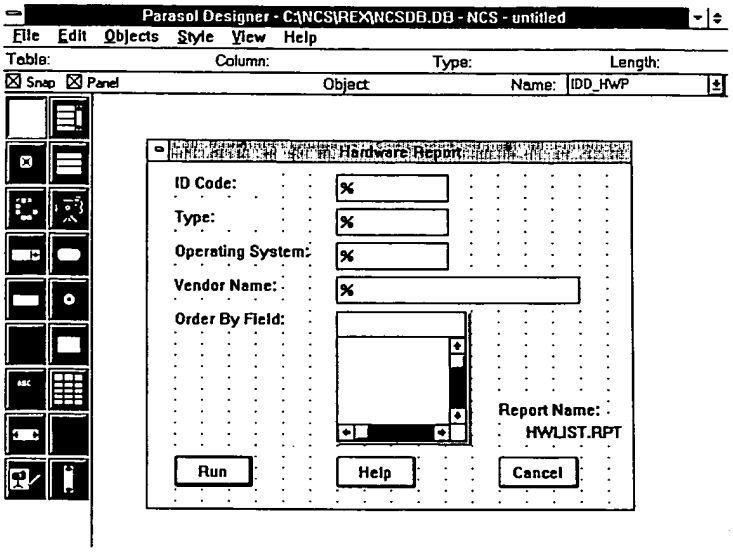

Figure 5-25: Report Dialog panel design.

After you design your report dialog panel, select the panel and double click with the right mouse button to access the detail specification dialog. On the first dialog, when you select the Report button, it will change to the details for a report dialog panel.

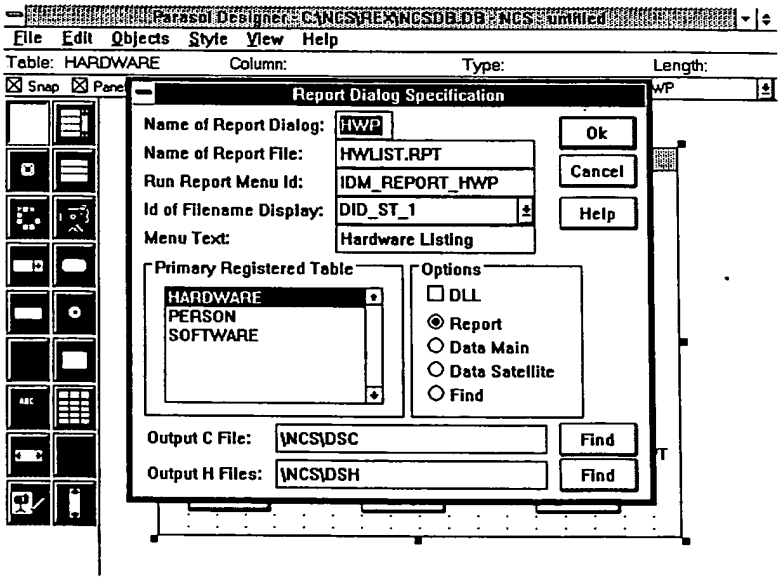

Figure 5-26: Report Specification dialog screen for Report panel - ReportSmith

Select a table name that this panel is connected to. This will fill in all the fields except the parameters as described below:

- Name of Report Object - Name of the object as specified in sections 3.4.1 & 3.4.3

For Database Manager:

- Name of Report File - Name of disk file the report will be printed to
- Name of Query in Database Manager - Name of report query in Query Manager
- Name of Form in Database Manager - Name of report form in Query Manager

For ReportSmith:

- Name of Report File - Name of ReportSmith report file on disk - Parasol Executive assumes the report file is in the current working directory when your application runs
- Run Report Menu ID - ID of menu that will run the report dialog
- Id of Filename Display - Report dialogs MUST contain a static text object that will display the report filename to the end-user.
- Menu Text - Text that appears on the Report menu of the application - The Source Code Generator will append the ellipse (...) characters to the menu file
- DLL - Check if the source code is to be put into a Dynamic Link Library. This is an advanced user option.

If default naming has been turned off the following parameters must be filled in.

- Output C File - Directory where "C" object is to be placed. Recommend using the \DSC directory of your project name.
- Output H files - Directory where "C" object headers are to be placed. Recommend using the \DSH directory of your project name.

When you have entered all the appropriate information in these specification dialogs for each panel, select the Ok button to save that information.

5.2.2 Create Controls and Assign Project Definitions Automatically

If you are using default naming conventions and want default objects on your dialog panels, you can simply use a shortcut method to design a dialog panel. This uses a special option on the File Menu called Table View. With you, you "drag and drop" objects from the data table definition in your database onto the dialog panel. After a dialog has been assigned a table name in section 5.2.1 above, the File - Table View menu becomes ungreyed. Select the File - Table View menu to access this automatic naming and design function.

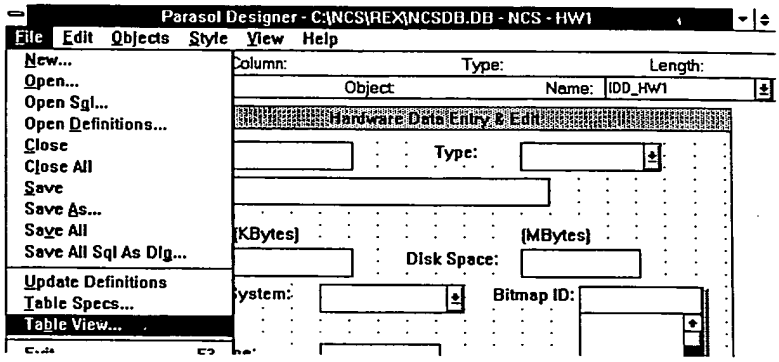

Figure 5-27: Table View option highlighted on the File Menu in Parasol Designer.

A window appears that shows the columns that exist in the database table.

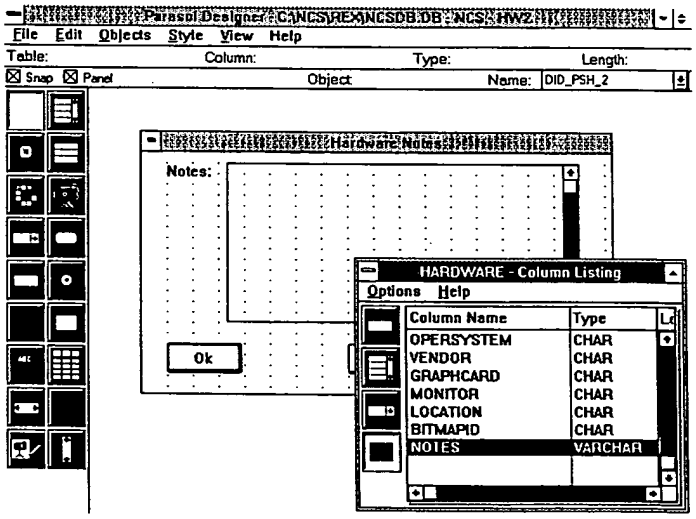

Figure 5-27: Table View Window for dragging columns onto the dialog.

Select the type of object you wish to associate with a particular column from the icon toolbar and then start a drag and drop operation by selecting the column name with the right mouse button. Hold the right mouse button down and move the cursor over to the dialog and release the mouse button where you want the object to be placed. The column name text will also automatically be place beside the object and the control linked to the column of the database table.

This is an automated method for placing objects onto dialog panels, and connecting them to the data tables in your database. See the next section to edit this link.

5.2.3 Assign Project Definitions to Dialog Controls

If you have used the automatic object assignment functions described in Section 5.2.2, these steps are unnecessary except to make change to an object in the panel.

After you have stored the panel specification in the database, you first select an object on the panel. Then, double click on the object with the RIGHT mouse button or push ctrl-spacebar and the Control Specification for that dialog box will appear.

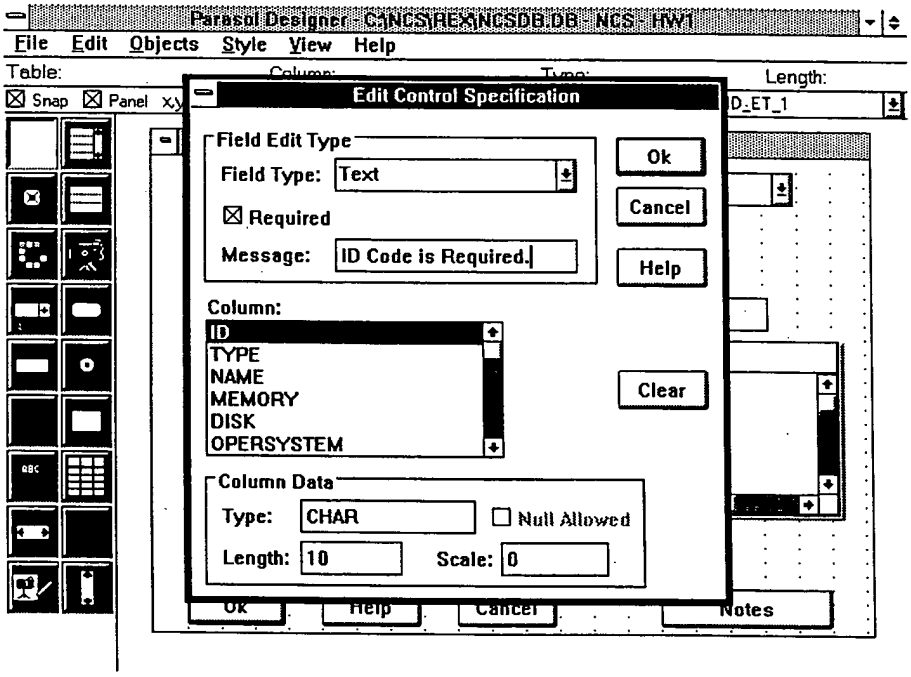

Figure 5-28: Edit Control Specification for a data entry field on the dialog.

This is done to every active dialog control that communicates with a database table. Select the object and double click with the right button or push ctrl-spacebar. Each control has a different dialog panel with the following possibilities:

Field Edit Type:

Field Type: This associates the dialog control with the field type used in Parasol Executive. If a column in a table was declared as DATE then this value must match the table definition. Warnings will appear in the compiler error log indicating different levels of indirection if incompatabilities exist.

Possible values are,

- TEXT Use with CHAR and VARCHAR column definitions
- DATE Use with DATE column definitions
- TIME Use with TIME column definitions
- TIME STAMP Use with TIMESTAMP column definitions
- MONEY Use with DECIMAL or MONEY column definitions
- MEDIUMINT Use with SMALLINT column definitions
- LARGEINT Use with INTEGER column definitions
- DOUBLE Use with DECIMAL column definitions

Message:

Message to display to the user if the required data has not been entered.

Column:

Name of the column to transfer data from and to the dialog control.

Column Data:

The column data shown at the bottom of the dialogs is the data from the database system catalog about the column in the table. If the check box "Nulls" is NOT checked your column does not allow NULL values and should be flagged as a required field. Do not flag columns as required in the find dialogs because here the user is not creating or editing the record, merely quering the table rows with an SQL where clause.

Fill Option:

The fill options appear for comboboxes and listboxes. Parasol will automatically load data from the PAR_SELECTION table (see Appendix A) into these control types. Another possibility is to load records from the PAR_BITMAP table (see Appendix A). Here we specify which one of these options we want to use. The selection TYPE value is the TYPE field in the PAR_SELECTION table. Only records of this type will be extracted from the table using a where clause.

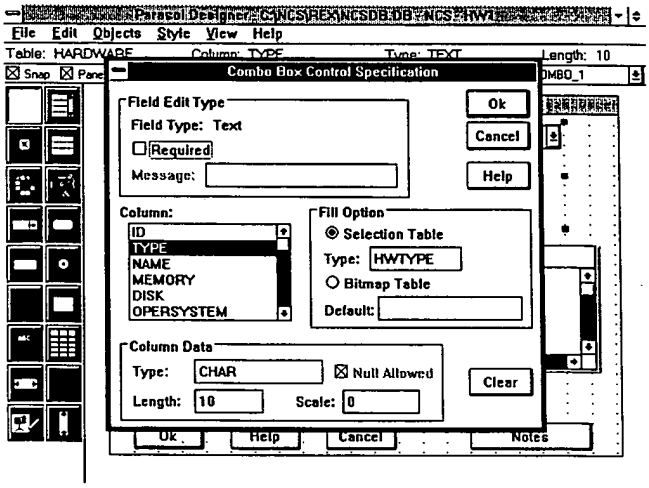

Figure 5-29: Combo Box Control Specification dialog box.

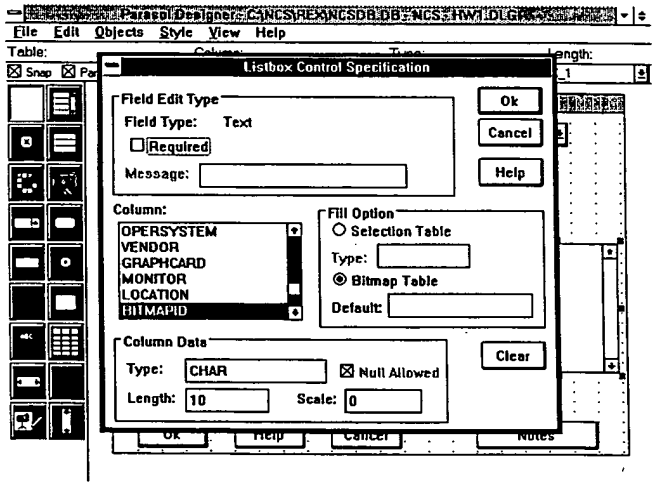

Figure 5-30: List Box Control Specification dialog box.

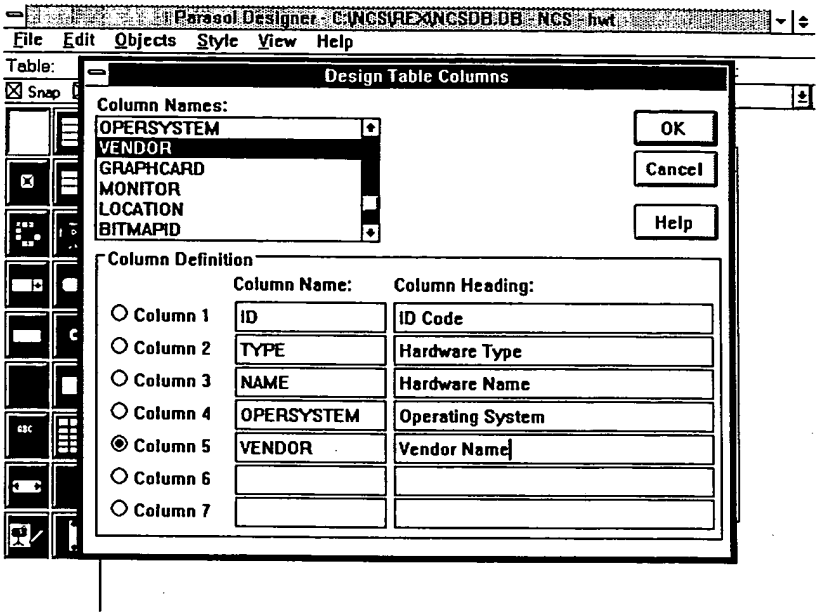

Figure 5-31: Define columns for a Table View screen.

When you have entered the information for each object on the dialog panel, select the Ok button on the Control Specification dialog box and return to the Parasol Designer main screen showing the panel you are working with. Make sure you use consecutive column numbers and don't leave any blanks (i.e. enter a column name in column 4 without a name and heading in column 3).

5.2.4 Assign Report Definitions to Dialog Controls

Report dialog panels have special characteristics because they must communicate with a report writer and a report specification file. However, the objects you place on a report dialog panel are the same as for other data entry dialogs. This is done to every active dialog control that communicates with a report writer.

Select the object and double click with the right button or push ctrl-spacebar. An Edit Control Specfication for each object will appear on the screen as follows:

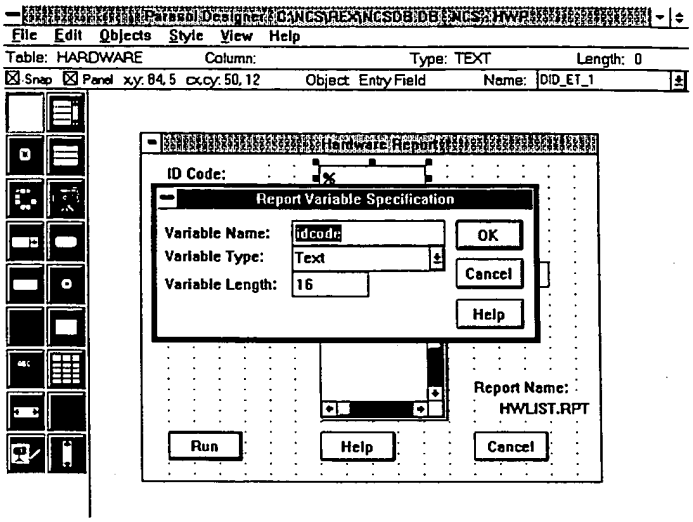

Figure 5-32: Edit Control Specification for a report variable field on the dialog.

Each control has a different dialog panel with the following possibilities:

Field Edit Type:

Field Type: This associates the dialog control with the field type used in Parasol Executive. If a variable in a report was declared as DATE then this value must match the report definition. The report writer will produce errors if incompatablities exist.

Possible values are,

- TEXT Use with string variables
- DATE Use with date variables
- TIME Use with time variables
- TIME STAMP Use with timestamp variables
- MEDIUMINT Use with integer variable
- LARGEINT Use with 4 byte integer variables

Variable Length:

Space that should be allocated to the variable.

Fill Option:

The fill options appear for comboboxes and listboxes. Two possibilities are available for filling combo and listboxes with data:

- OrderBy Field that Loads Column Names -Select a table name and Parasol Executive will load a list of all the columns in that table (for use in the Order By clause of SQL).
- Selection Table - Parasol Executive will automatically load data from the PAR_SELECTION table (see Appendix A) into these control types. Here we specify which one of these options we want to use. The selection TYPE value is the TYPE field in the PAR_SELECTION table. Only records of this type will be extracted from the table using a where clause.

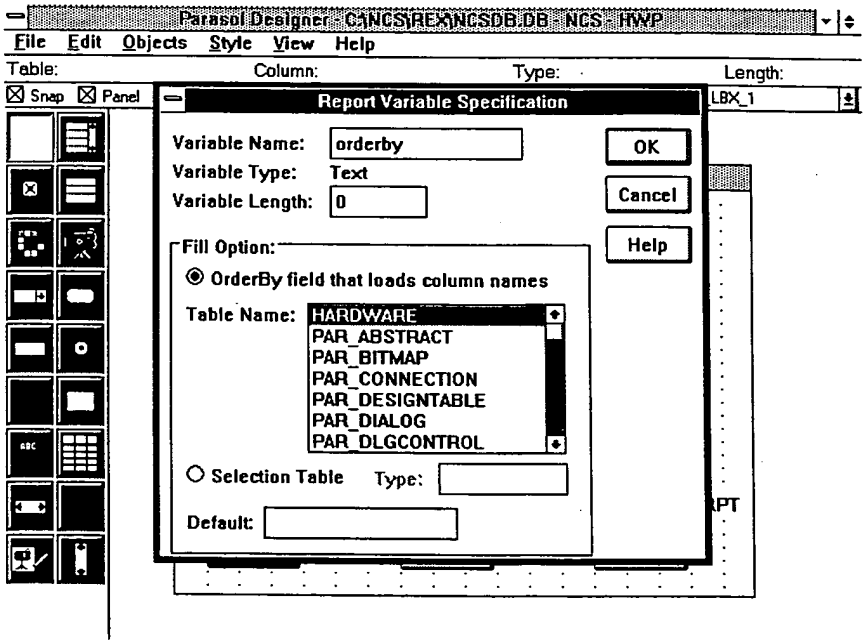

Figure 5-32: Report Variable Specification dialog box for the Order By criteria.

5.2.5 Save the Dialog Panels

When you have completed all the steps for designing a dialog box, select the Save option of the File menu. Since these dialog panels are now database records, they will be updated in the database.

Stick to the naming conventions discussed in section 3.4.1 when naming dialogs. The dialogs should always be named the same as your "C" object that uses the dialog.

5.2.6 Change Dialogs

Dialog panels can be re-read from the database and revised. If controls are only changed graphically then the Parasol Source Code Generator will not need to be rerun.

5.2.7 Backup Dialog Panel Definitions

The dialog panel definitions are stored in the database tables PAR_DIALOG and PAR_DLGCONTROL (see Appendix A). These tables should be exported to disk files using the export utility of your SQL datbase. In this way as you go back and recreate the database interactively during the project life-cycle, you can reload the table with your specification data.

When you have specified and saved all the dialog designs, you will be ready to start generating the source code for them, and then compile and link them into your customized Parasol application.

Chapter 6 - Generate Code

Now that you have specified your application, it is time to generate the C code that represents all the objects you have designed. This phase has the following steps that you execute in the Parasol Source Code Generator. They are:

- Generate code for the Application objects
- Generate code for the Dialog objects
- Generate code for the Database Access objects
- Generate the Default Menu resource file
- Generate the Library Definitions file
- Generate the Utility files: compiler setup, make and link
- Populate the PAR_ITEMTYPE table with Design Types

All the steps above are automatically performed when you select the menu option to Generate All objects in your application. These are described separately to demonstrate the comprehensive functions of the Source Code Generator.

These are listed on the file menu of the startup screen for the Source Code Generator.

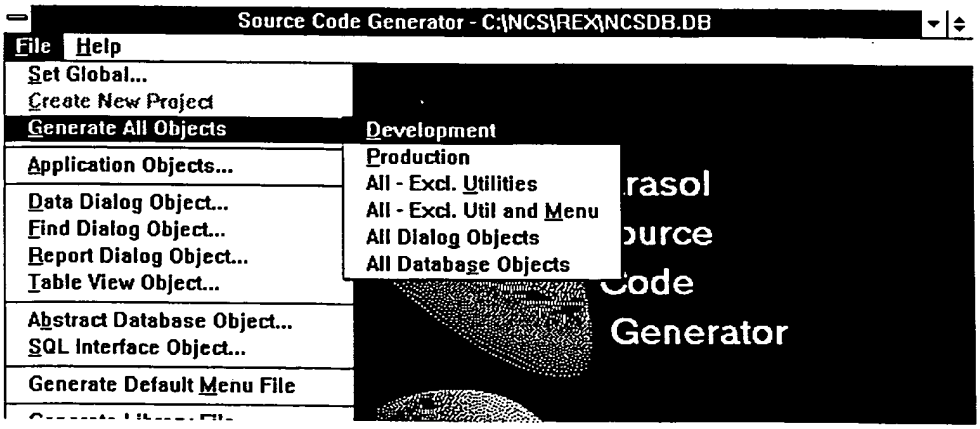

Figure 6-1: Source Code Generator startup screen with File Menu and the Generate All Objects option highlighted.

This chapter describes the procedure for generating the code for each of the objects required for your application. Then, in the next phase, you will compile and link them into a complete application program.

6.1 Details about the Source Code Generator

This section will explain some of the global features of the Parasol Source Code Generator.

6.1.1 Parasol Source Code Generator - Template Files

The Parasol Source Code Generator makes use of template files to create the source code. These are all stored in a subdirectory called TPL in the PARASOL directory where you installed Parasol Developer.

You should modify the copyright template file so that your copyright appears on the source code you create. This file is called COPYRITE.INC and contains the text for a Copyright Notice. Change this for your own notice as you wish. You can review other template files to see if there are any further changes you might want to make. There are over 100 files altogether, so be careful not to change any drastically since it may cause errors in the code generation procedures.

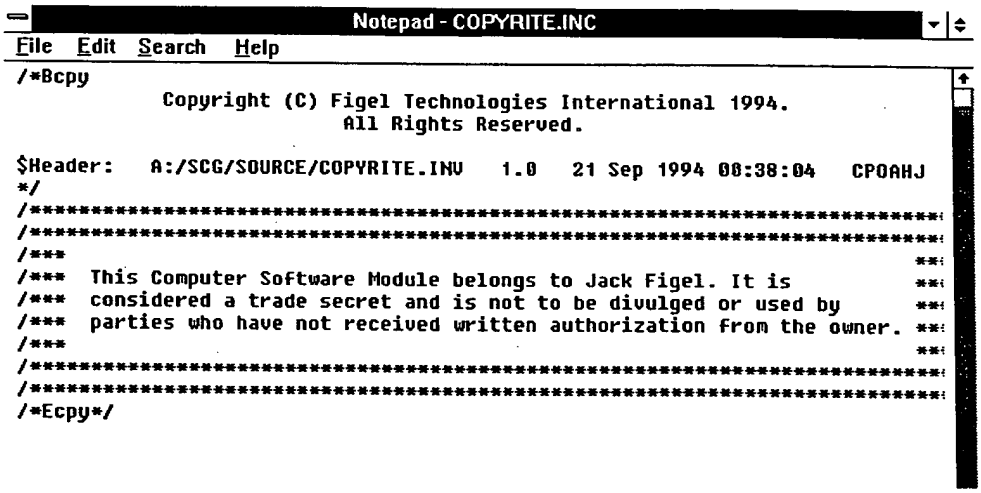

Figure 6-2: Sample template file showing copyright notice.

6.1.2 Parasol Source Code Generator - Time Stamps

The objects that have their specification in the database have a time stamp indicating the last time they were updated using Parasol Designer. The Source Code Generator uses this information to check if the source code on the disk is consistent with the specification or not. Therefore, any of the menu commands that Generate All will only update objects based on this time stamp. If you wish to force generation the time stamp must be updated by saving the object to the database using Parasol Designer.

6.1.3 Parasol Source Code Generator - Validation Log

The Source Code Generator validates the design specification for each object that is created. If you use any of the Generate All Objects submenus a validation log is created in the \CL directory of your project directory. The log file name is SCG.LOG and a search on the string "Warning" or "Error" is a quick way to scan the log for any changes that are required to the specification.

6.1.4 Parasol Source Code Generator - Source Code Markup Merging

The Source Code Generator has the abitity to merge changes to the source code should you want to make manual changes to the objects created by the generator. This applies to ALL file types that are written by the Source Code Generator.

The Source Code Generator merges any changes in the files with the new output. Mark-up consists of the following:

Block merge:	SCG_START SCG_END
Merge line:	SCG_KEEP 1 or SCG_KEEP
Merge lines:	SCG_KEEP n - where n is the number of lines to merge

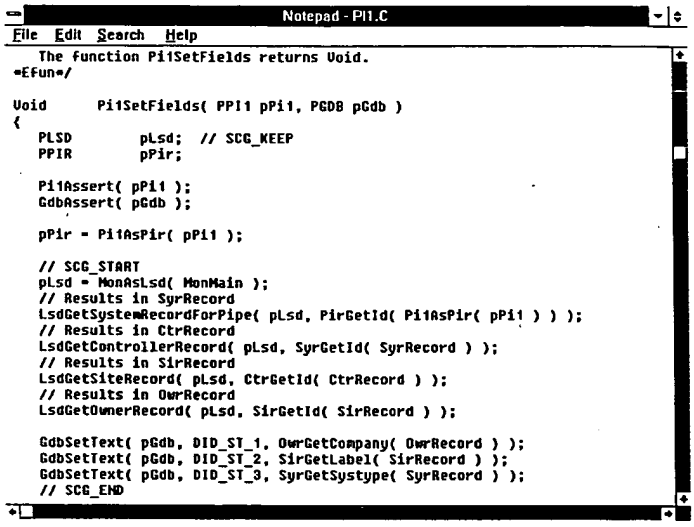

Figure 6-3: Sample object file showing source code markups.

6.2 Generate Source Code for Application Objects

Application objects are "system" level objects that make your Parasol Executive application operate. You generate these from the Source Code Generator. Then you select the Application Objects option of the Source Code Generator.

This section describes the various code generation steps to create the Application Objects.

6.2.1 Generate System Objects

There are four system objects that you generate called Application Objects. These are:

- Application Object
- Application Menu Manager Object
- Application Record Manager Object
- Tree Building Object

Run the Parasol Source Code Generator and select the File Menu and click on the option Application Objects.

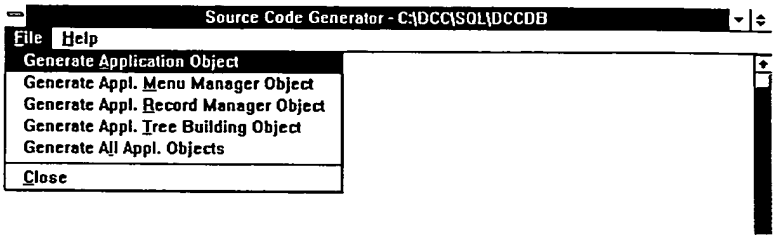

Figure 6-4: Application Objects screen with generation options shown.

Select each of the generate options in turn for the four system objects. This process will create those objects which are system related and necessary for your application. They are:

- Generate Application Object - the topmost object in the hierarchy of objects and the object that inherits Parasol.
- Generate Application Menu Manager Object - the methods in this file are called when the user touches your menu item.
- Generate Application Record Manager Object - this object holds one global variable for each table type in your application. The variable is one instance of an abstract record of the database table.
- Generate Application Tree Building Object - the methods in this object control the display in Parasol's outline or 3D.
- Generate All Objects - this option will generate all system objects at once.

As each object is generated, a message will appear on the screen.

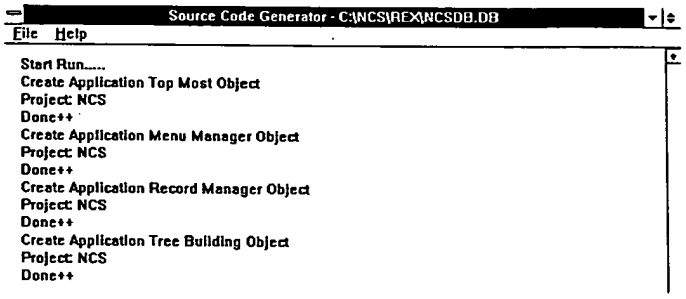

Figure 6-5: Messages from generating all system objects.

When you are finished with all objects, select the Close option of the File Menu to return to the main Source Code Generator screen

6.3 Generate Source Code for Dialog Panels

This step will generate the source code for all the dialog panels you defined in Parasol Designer. Like the other objects, you first select each one and then generate its code. Or you can select the Generate All option to make them all at once.

6.3.1 Generate Data Dialog Objects

First select the Data Dialog Object option of the File menu in the Source Code Generator. Then pick Select Data Dialog Object from the File menu in the current screen.

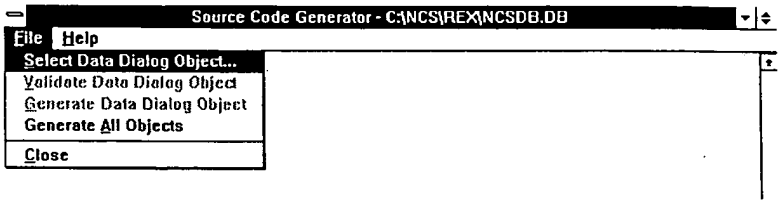

Figure 6-6: Data Dialog Object option highlighted on the File Menu.

When you select that option, a dialog box will appear listing some of the currently specified data dialogs stored in the database. Pick the data dialog you want to generate by clicking on it. When you select it, all the associated definition information will also appear in the dialog box to confirm the details for that data dialog.

You should validate all Data Dialog objects before you start the generation of them.

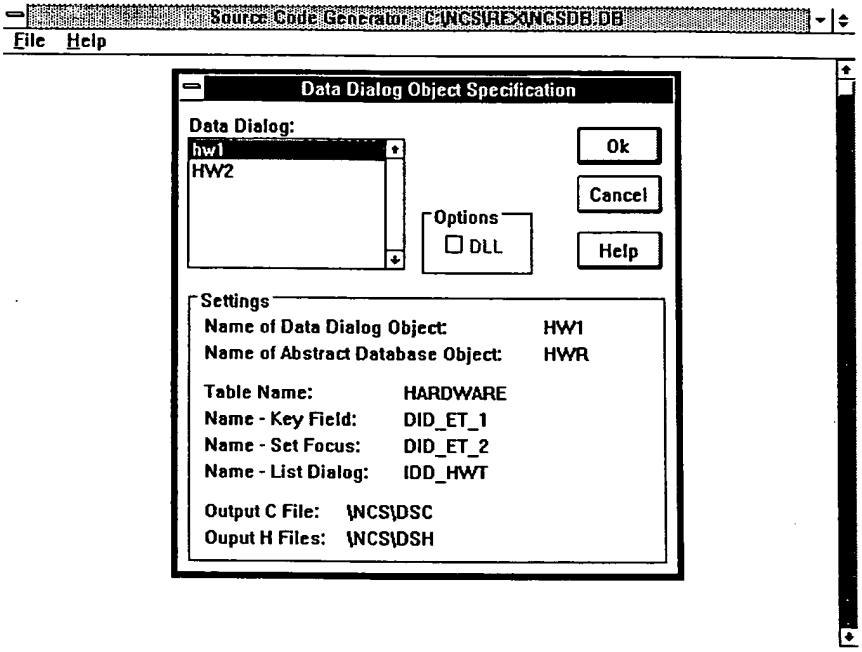

Figure 6-7: Data Dialog Object Specification dialog box with the Hardware table selected [HW1].

After you select it, click on Ok and return to the generate screen. Now the other File Menu options are available -- Validate, Generate and Generate All.

If you just want to validate the data dialog selected, select that menu item. This will run a report validating your specification.

If you find any errors, make the changes in Parasol Designer and rerun the validation. You can keep the Source Code Generator and Parasol Designer running at the same time if the database allows two sessions at once, and just switch between programs.

Once the validation report has no errors, you are ready to generate the object. Select the Generate Data Dialog Object option of the File Menu. This option will first validate the specification, and then generate the "C" code object for this dialog and store the resulting files in the directories named in the specification process of Parasol Designer described in section 5.2 (recommend DSC and DSH subdirectories of your project directory).

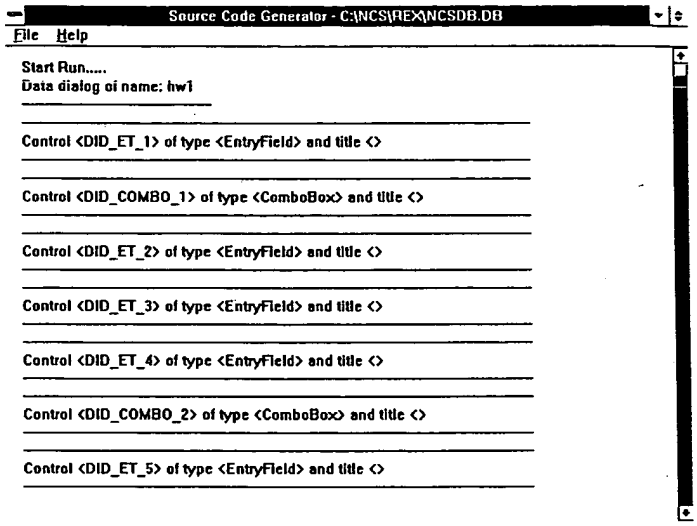

Figure 6-8: Validation report for the Hardware Data table.

The validation report will first appear, and then the listing of objects used in the generation of the source code file. You can review this listing by using the Page Down key. The last statement in the listing is the word "Done++" which indicates a successful completion.

If you have a large number of Data Dialogs to generate, you may want to validate each first, then use the Generate All option on the File menu to generate all the Data Dialog objects at once. When you are finished, select the Close option and return to the main screen of Parasol Source Code Generator.

During any of the generation operations, you can click on the File menu option to halt the processing.

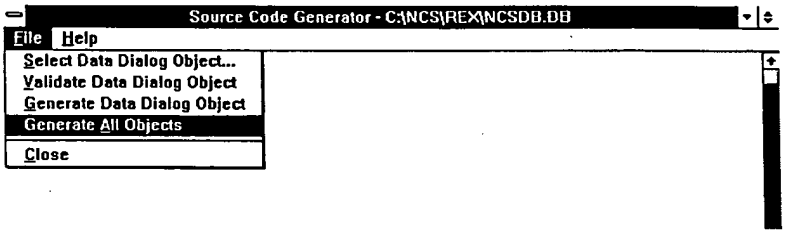

Figure 6-9: File Menu with the Generate All Objects option highlighted.

When you run these processes, you may encounter possible error messages. They may be:

Output file cannot be opened:

Meaning - A open file failed. Check name of file and directory.

Template file cannot be opened:

Meaning - A template file could not be found. Check name of the directory defined in the Set Global dialog.

On the panel;

- Error - no table name
- Error - no name for data dialog object
- Error - no name for abstract database record object
- Error - no name given for control that is the primary key field
- Error - no name given for control that should have the focus on data change
- Error - no name given for the table view dialog that is associated

On the controls;

- Warning - Not supported
- Warning - No user message installed for control
- Error - Control has no SQL column name
- Error - Control has UNKNOWN edit type
- Error - Control must be given a name other than DID_NULL
- Error - Control must be of edit type - Text
- Error - Combo box filled from PAR_SELECTION table must have a type
- Error - Combo box must have a type (Selection Table or Bitmap)
- Error - Listbox filled from PAR_SELECTION table must have a type value
- Error - Listbox must have a type (Selection Table or Bitmap)

If any error or warning messages appear, make note of the error and the object that you were trying to generate. Then return to Parasol Designer to make changes to that object as necessary to correct the error. Return to the Code Generator and attempt to generate the object again.

6.3.2 Generate Find Dialog Objects

First select the Find Dialog Object option of the File menu in the Source Code Generator. Then pick Select File Dialog Object from the File menu in the current screen.

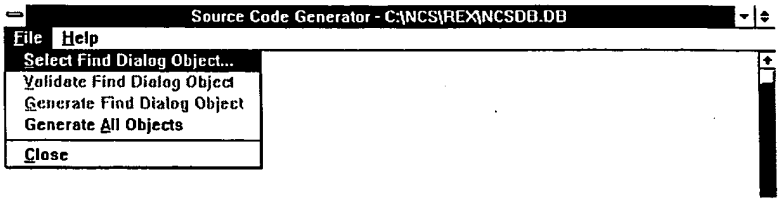

Figure 6-10: Select Find Dialog Object option highlighted on the File Menu.

When you select that option, a dialog box will appear listing all the currently specified find dialogs stored in the database. Pick the find dialog you want to generate by clicking on it. When you select it, all the associated definition information will also appear in the dialog box to confirm the details for that find dialog.

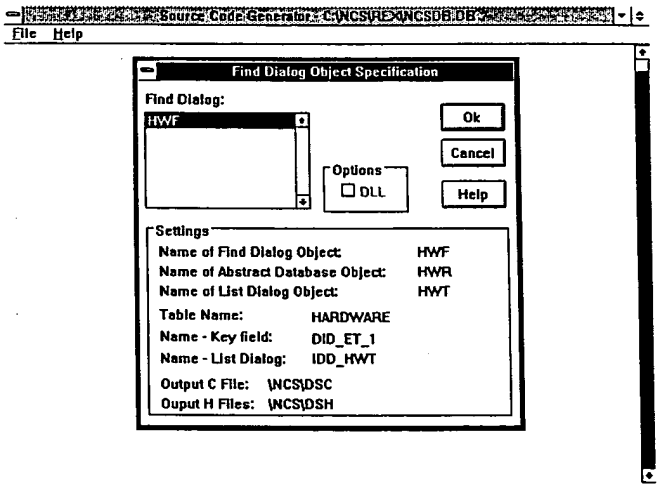

Figure 6-11: Find Dialog Object Specification dialog box with the Hardware table selected [HWF].

After you select it, click on Ok and return to the generate screen. Now the other File Menu options are available -- Validate, Generate and Generate All.

If you just want to validate the find dialog selected, select that menu item. This will run a report validating your specification. If you find any errors, make the changes in Parasol Designer and rerun the validation. You can keep the Source Code Generator and Parasol Designer running at the same time if the database allows two sessions at once, and just switch between programs.

Once the validation report has no errors, you are ready to generate the object. Select the Generate Find Dialog Object option of the File Menu. This option will first validate the specification, and then generate the "C" code object for this dialog and store the resulting files in the directories named in the specification process in Parasol Designer described in section 5.2 (recommend DSC and DSH subdirectories of your project directory).

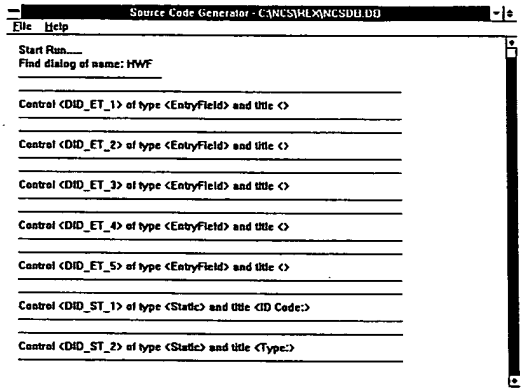

Figure 6-12: Validation report for the Hardware Find table.

The validation report will first appear, and then the listing of objects used in the generation of the source code file. You can review this listing by using the Page Down key. The last statement in the listing is the word "Done++" which indicates a successful completion.

If you have a large number of Find Dialogs to generate, you may want to validate each first, then use the Generate All option on the File menu to generate all the Find Dialog objects at once. When you are finished, select the Close option and return to the main screen of Parasol Source Code Generator.

During any of the generation operations, you can click on the File menu option to halt the processing.

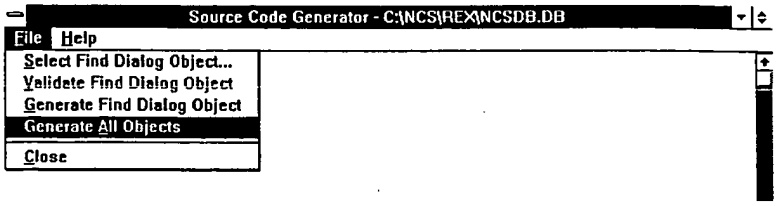

Figure 6-13: File Menu with the Generate All Objects option highlighted.

When you run these processes, you may encounter some possible error messages. They are:

Output file cannot be opened:

Meaning - A open file failed. Check name of file and directory.

Template file cannot be opened:

Meaning - A template file could not be found. Check name of the directory defined in the Set Global dialog.

On the panel;

- Error - no table name
- Error - no name for find dialog object
- Error - no name for abstract database record object
- Error - no name for table view dialog object
- Error - no name given for control that is the primary key field
- Error - no name given for the table view dialog that is associated
- Error - No key found for table - assign one column to be NOT NULL
- Error - First NOT NULL column name of table does not match any control column name
- Error - Name of key field does not match column name that is the first NOT NULL column name of table

On the controls;

- Warning - Not supported

- Warning - No user message installed for control
- Error - Control has no SQL column name
- Error - Control must be given a name other than DID_NULL
- Error - Control has UNKNOWN edit type
- Error - Control must be of edit type - Text
- Error - Combo box filled from PAR_SELECTION table must have a type value
- Error - Combo box must have a type (Selection Table or Bitmap)
- Error - Listbox filled from PAR_SELECTION table must have a type value
- Error - Listbox must have a type (Selection Table or Bitmap)

If any error or warning messages appear, make note of the error and the object that you were trying to generate. Then return to Parasol Designer to make changes to that object as necessary to correct the error. Return to the Code Generator and attempt to generate the object again.

6.3.3 Generate Table View Dialog Objects

First select the Table View Object option of the File menu in the Source Code Generator. Then pick Select Table View Object from the File menu in the current screen.

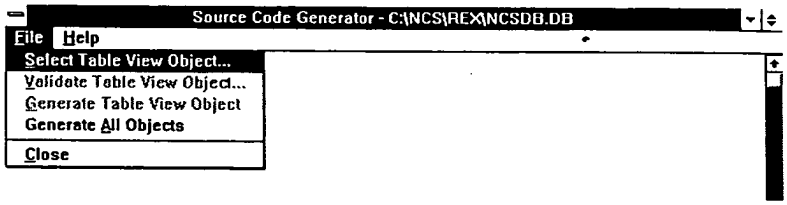

Figure 6-14: Select Table View Object option highlighted on the File Menu.

When you select that option, a dialog box will appear listing all the currently specified table views stored in the database. Pick the table view you want to generate by clicking on it. When you select it, all the associated definition information will also appear in the dialog box to confirm the details for that table view.

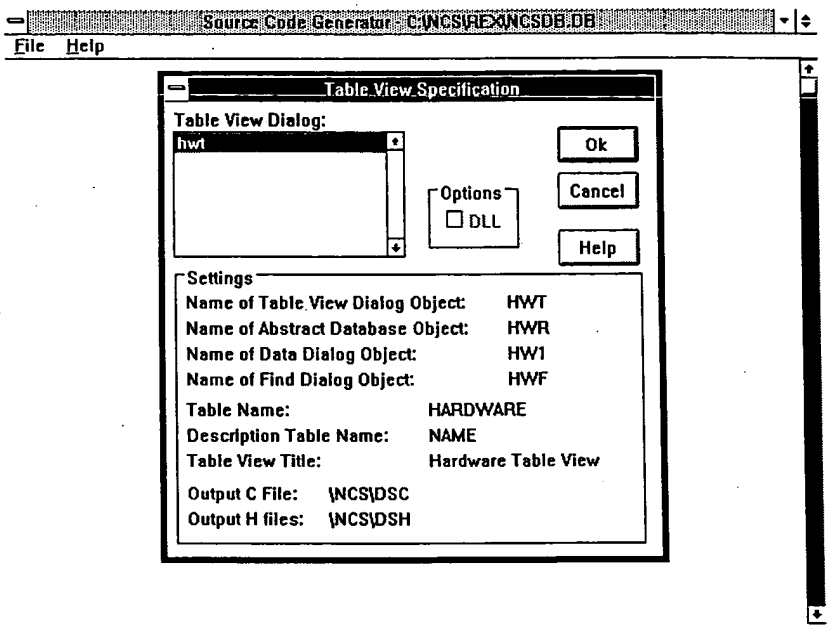

Figure 6-15: Table View Object Specification dialog box with the Hardware table selected [HWT].

After you select it, click on Ok and return to the generate screen. Now the other File Menu options are available -- Validate , Generate and Generate All.

If you just want to validate the table view dialog selected, select that menu item. This will run a short report validating your specification. If you find any errors, make the changes in Parasol Designer and rerun the validation. You can keep the Source Code Generator and Parasol Designer running at the same time if the database allows two sessions at once, and just switch between programs.

Once the validation report has no errors, you are ready to generate the object. Select the Generate Table View Object option of the File Menu. This option will first validate the specification, and then generate the "C" code object for this dialog and then store the resulting files in the directories named in the specification process in Parasol Designer described in section 5.2 (recommend DSC and DSH subdirectories of your project directory).

The validation report will first appear, and then the listing of objects used in the generation of the source code file. The last statement in the listing is the word "Done++" which indicates a successful completion.

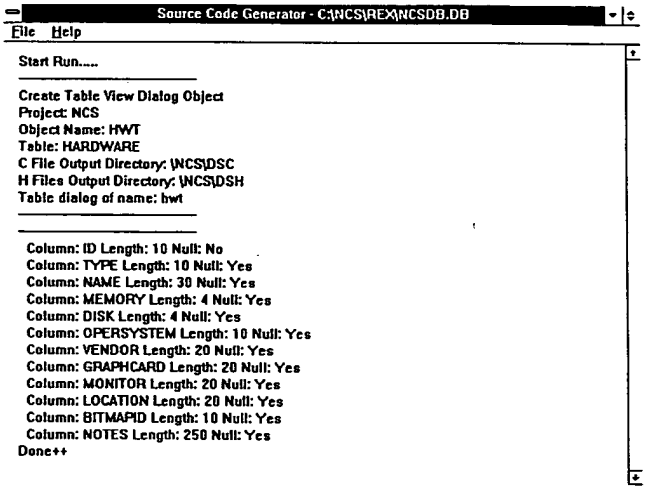

Figure 6-16: Messages from generating the Hardware Table View.

If you have a large number of Table View Dialogs to generate, you may want to validate each first, then use the Generate All option on the File menu to generate all the Table View Dialog objects at once. When you are finished, select the Close option and return to the main screen of Parasol Source Code Generator.

During any of the generation operations, you can click on the File menu option to halt the processing.

When you run these processes, you may encounter some possible error messages. They are:

Output file cannot be opened:

Meaning - A open file failed. Check name of file and directory.

Template file cannot be opened:

Meaning - A template file could not be found. Check name of the directory defined in the Set Global dialog.

No reference could be found to column [] in your specification

The column [] will default to type TEXT. A column has been defined in the Table View columns that is not used anywhere else in the specification and therefore the data is assumed to be of type TEXT.

On the panel;

- Error - no table name
- Error - no name for table dialog object
- Error - no name for abstract database record object
- Error - no name for data dialog object
- Error - no name for find dialog object
- Error - Table view object has no title

If any error or warning messages appear, make note of the error and the object that you were trying to generate. Then return to Parasol Designer to make changes to that object as necessary to correct the error. Return to the Code Generator and attempt to generate the object again.

6.3.4 Generate Report Dialog Objects

First select the Report Dialog Object option of the File menu in the Source Code Generator. Then pick Select Report View Object from the File menu in the current screen.

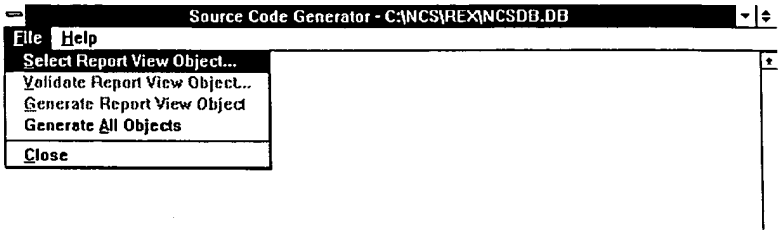

Figure 6-17: Select Report View Object option highlighted on the File Menu.

When you select that option, a dialog box will appear listing all the currently specified report dialogs stored in the database. Pick the report dialog you want to generate by clicking on it. When you select it, all the associated definition information will also appear in the dialog box to confirm the details for that report dialog.

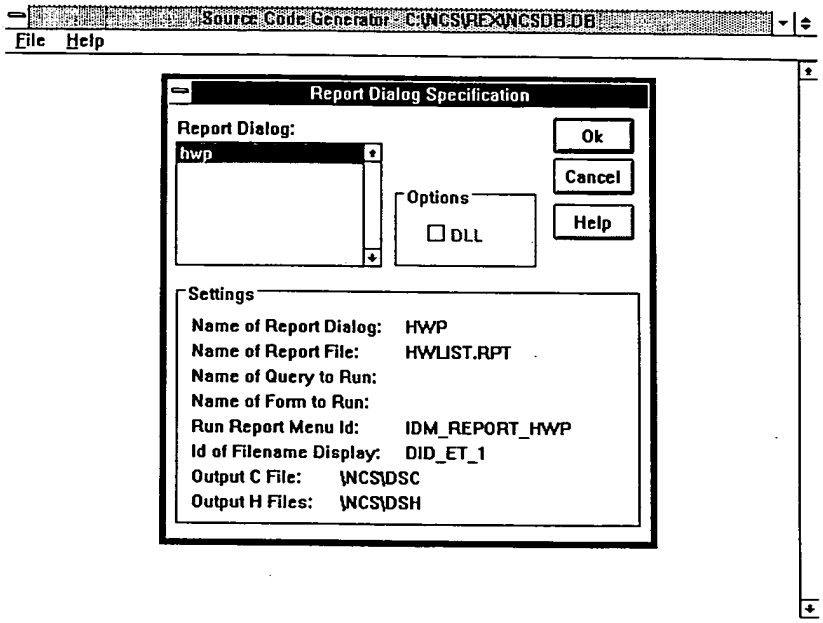

Figure 6-18: Report Object Specification dialog box with the Hardware report selected [HWP].

After you select it, click on Ok and return to the generate screen. Now the other File Menu options are available -- Validate , Generate and Generate All.

If you just want to validate the report dialog selected, select that menu item. This will run a short report validating your specification. If you find any errors, make the changes in Parasol Designer and rerun the validation. You can keep the Source Code Generator and Parasol Designer running at the same time if the database allows two sessions at once, and just switch between programs.

Once the validation report has no errors, you are ready to generate the object. Select the Generate Report View Object option of the File Menu. This option will first validate the specification, and then generate the "C" code object for this dialog and then store the resulting files in the directories named in the specification process in Parasol Designer described in section 5.2 (recommend DSC and DSH subdirectories of your project directory).

The validation report will first appear, and then the listing of objects used in the generation of the source code file. The last statement in the listing is the word "Done++" which indicates a successful completion.

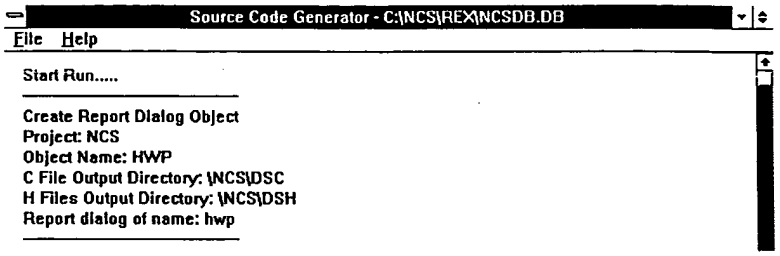

Figure 6-19: Messages from generating the Hardware Report View Dialog.

If you have a large number of Report Dialogs to generate, you may want to validate each first, then use the Generate All option on the File menu to generate all the Report Dialog objects at once. When you are finished, select the Close option and return to the main screen of Parasol Source Code Generator.

During any of the generation operations, you can click on the File menu option to halt the processing.

When you run these processes, you may encounter some possible error messages. They are:

Output file cannot be opened:

Meaning - A open file failed. Check name of file and directory.

Template file cannot be opened:

Meaning - A template file could not be found. Check name of the directory defined in the Set Global dialog.

On the panel;

- Error - no name for report dialog object
- Error - no filename given for report
- Error - no name given for report query
- Error - no name given for report form
- Error - no Id given for report menu
- Error - no Id given for report filename display on dialog

If any error or warning messages appear, make note of the error and the object that you were trying to generate. Then return to Parasol Designer to make changes to that object as necessary to correct the error. Return to the Code Generator and attempt to generate the object again.

6.4 Generate Source Code for Database Objects

There are two special objects for each table in your application that you must generate. These are the Abstract Database Record Object and the SQL Interface Database Object. This section will describe how to generate these two objects for your application.

6.4.1 Generate Abstract Database Record Object

First select the Abstract Database Object option of the File menu in the Source Code Generator. Then pic Select Abstract Database Object from the File menu in the current screen.

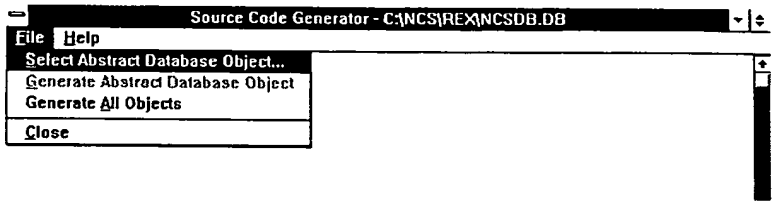

Figure 6-20: Select Abstract Database Object option highlighted on the File Menu.

When you select that option, a dialog box will appear listing all the currently specified abstract record definitions stored in the database. Pick the abstract record definition you want to generate by clicking on it. When you select it, all the associated definition information will also appear in the dialog box to confirm the details for that abstract record definition.

After you select it, click on Ok and return to the generate screen. Now the other File Menu options are available -- Validate, Generate and Generate All.

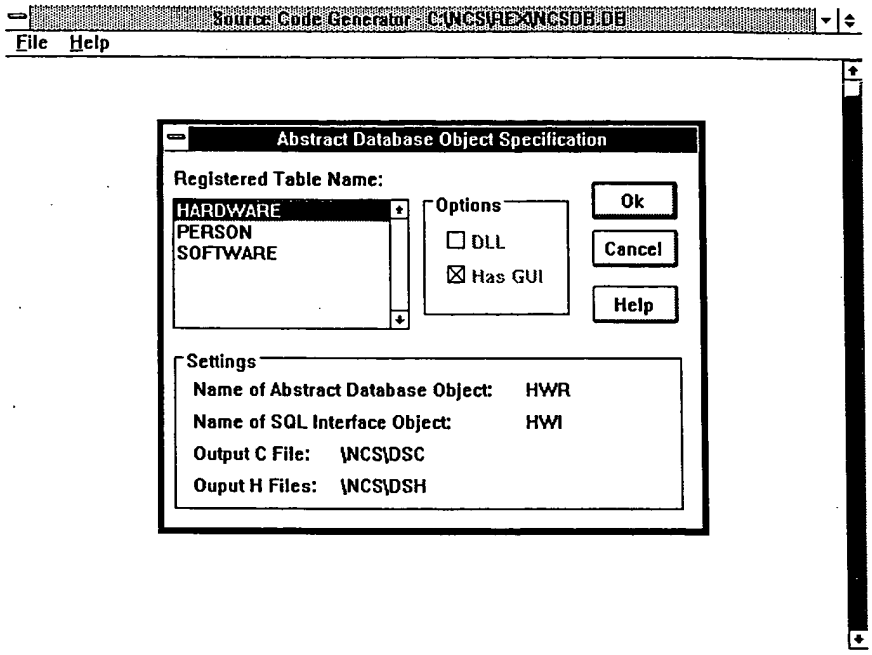

Figure 6-21: Abstract Database Object Specification dialog box with the Hardware table selected [HWR].

After you enter this information, click on Ok and return to the generate screen. Now the other File Menu options is available -- Generate and Generate All.

Select the Generate Abstract Database Object option of the File Menu. This option will generate the C code object for this Abstract Database Record object. The resulting files will be stored in the directories named in Parasol Designer Table Registration (recommend DSC and DSH subdirectories of your project directory).

The listing of objects used in the generation of the source code file will appear on the screen. The last statement in the listing is the word "Done++" which indicates a successful completion. When you are finished, select the Close option and return to the main screen of Parasol Source Code Generator.

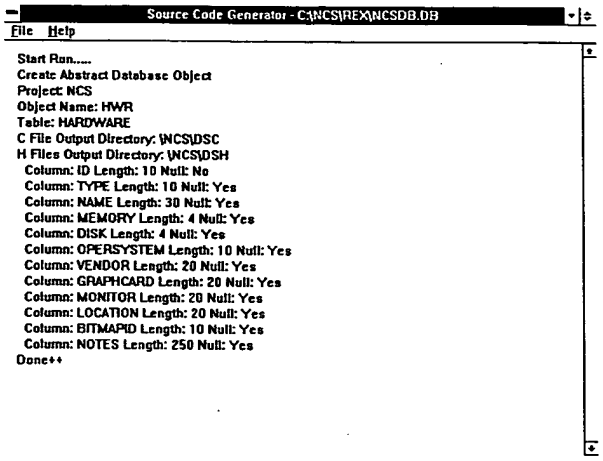

Figure 6-22: Messages from generating the Hardware Abstract Database Record.

When you run these processes, you may encounter some possible error messages. They are:

Output file cannot be opened:

Meaning - A open file failed. Check name of file and directory.

Error in template H file - cannot find reference to check word in C structure

Meaning - The template file ADOH.INC has been altered incorrectly. Replace the file with the file included with Parasol Developer. When modifying the file do not remove the reference to the check word in the C structure template.

Template file cannot be opened:

Meaning - A template file could not be found. Check name of the directory defined in the Set Global dialog.

No key found for table - assign one column to be NOT NULL

Meaning - The database table has no column name that can be identified as the key field. See section 3.3.2.

If any error or warning messages appear, make note of the error and the object that you were trying to generate. Then return to Parasol Designer to make changes to that object as necessary to correct the error. Return to the Code Generator and attempt to generate the object again.

6.4.2 Generate SQL Interface Database Object

First select the SQL Interface Object option of the File menu in the Source Code Generator. Then pick Select SQL Interface Object from the File menu in the current screen.

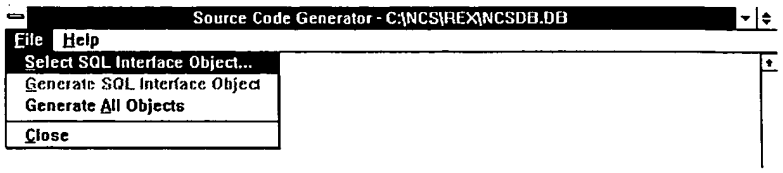

Figure 6-23: Select SQL Interface Object option highlighted on the File Menu.

When you select that option, a dialog box will appear listing all the currently specified SQL Interface definitions stored in the database. Pick the SQL Interface definition you want to generate by clicking on it. When you select it, all the associated definition information will also appear in the dialog box to confirm the details for that SQL Interface definition.

After you select it, click on Ok and return to the generate screen. Now the other File Menu options are available -- Generate and Generate All.

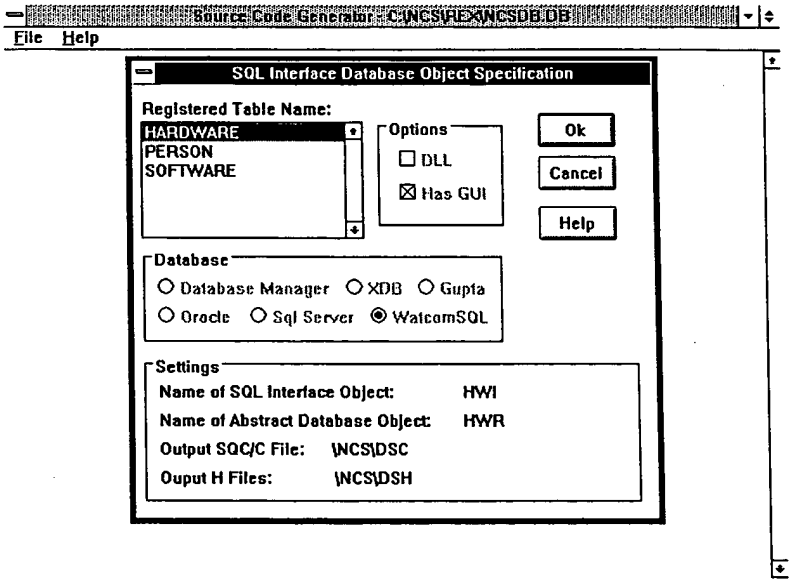

Figure 6-24: SQL Interface Object Specification dialog box with the Hardware table selected [HWI].

Select the Generate SQL Interface Object option of the File Menu. This option will generate the SQL preprocessor code in files with the extension .SQL (SqlBase does not have a preprocessor and the C file is generated directly). The resulting files will be stored in the directories named in Parasol Designer earlier (recommend DSC and DSH subdirectories of your project directory).

The listing of objects used in the generation of the source code file will appear on the screen. The last statement in the listing is the word "Done++" which indicates a successful completion. When you are finished, select the Close option and return to the main screen of Parasol Source Code Generator.

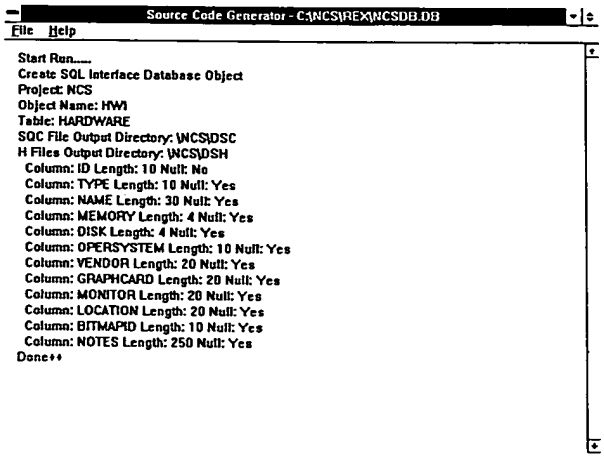

Figure 6-25: Messages from generating the Hardware SQL Interface Object.

When you run these processes, you may encounter some possible error messages. They are:

Output file cannot be opened:

Meaning - A open file failed. Check name of file and directory.

Template file cannot be opened:

Meaning - A template file could not be found. Check name of the directory defined in the Set Global dialog.

No primary key was found for this table:

Meaning - The database table has no column name that can identified as the key field. See section 3.3.2.

If any error or warning messages appear, make note of the error and the object that you were trying to generate. Then return to Parasol Designer to make changes to the parameters to correct the error. Rerun the Code Generator and attempt to generate the object again.

The Parasol Code Generator will automatically keep track of which objects have changed since the last code generation. So, all you really need to use the Generate All option on the main file menu of the Parasol Code Generator, and this will automatically generate any objects with corrections or changes made to them.

Also, under the Generate All option, you can select either Development or Production versions of code. The Development version is larger because it includes extra debugging code, where the Production version of the code is streamlined for production environments.

Chapter 7 - Compile and Link

7.1 Introduction

This is almost the final phase of building your application. Once you have generated all the C source code objects for your application, you are ready to compile and link them together to form an executable program file. First, there is some customizing that you can do before you start the compilation procedure. This involves the application logo, icon and accelerator table. This chapter starts with those items, and then describes the process for making your application program.

7.2 Create Windows or OS/2 Command Line Icon for Project

In order for the compiler and linker to find files you must create a project icon that uses a Windows or OS/2 icon to run a DOS or OS/2.

When you created the project with the Parasol Source Code Generator, it created a directory \CMD in your project directory XYZ.

In this directory you will find the following files:

- M.abc Command file to run development make file
- MP.abc Command file to run production make file
- XYZDLL.abc Command file to create project resource dynamic link library
- SETUP.abc Command file to setup project environment

where **abc** is CMD for OS/2 command files or BAT for DOS batch files.

If you used the Generate All command of the Source Code Generator the file SETUP.CMD in OS/2 or SETUP.BAT in Windows will have been written for you. If not, you can use the Source Code Generator menu command Utilities - Setup File to create this file from your definitions in the Set Global menu command. This file reflects the environment on your computer for this project. See section 7.4.1 for the difference between production and development

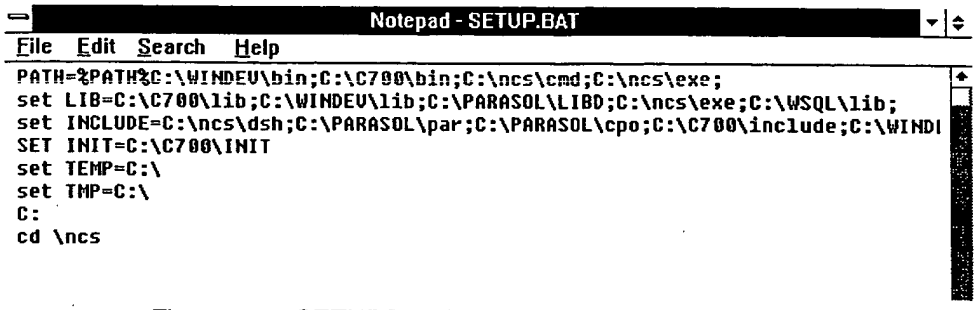

Figure 7-1: SETUP.BAT for the NCS example application.

For OS/2:

In order to run the Parasol Developer "make" file, you must run the file SETUP.CMD so as to setup the environment variables for compilation. We recommend making an OS/2 full-screen icon that runs this procedure at startup. This can achieved by creating a program item where the program file name is "*" and the parameters are **/K "%" c:\xyz\cmd\setup.cmd** where xyz is the project name. Also change the c drive letter to be the drive where you have created the project.

For Windows:

In order to run the Parasol Developer "make" file, you must run the SETUP.BAT so as to setup the environment variables for compilation. We recommend making a DOS full-screen icon with the following properties:

- Description: Give it the name of your project
- Command Line: [C7PRMT.PIF]
- Working Directory: [c:\xyz\cmd] where xyz is the project name in Parasol

Also, create a PIF file named C7PRMT.PIF with the following properties:

- Program Filename: COMMAND.COM
- Window Title: MS-DOS Prompt
- Optional Parameters: /E:2048
  (this sets enough space for the environmental variables)

After you run the Windows icon for the project, execute the SETUP.BAT file to setup the environmental variables for compilation.

7.3 Application Logo/Custom Icon/Accelerator Table

When you created the project with the Parasol Source Code Generator it created a directory \DSR and a directory \DSI. The \DSR directory holds the application resource files and the \DSI directory holds the application icons, pointers and bitmaps.

7.3.1 Insert Application Logo

In the \DSI directory you will find a bitmap called PARASOL.BMP. This is the default application logo bitmap. You can change this default bitmap by editing the file in your \DSR directory named XYZDLL.RC where xyz is the name of your project. Change the line

BITMAP IDB_LOGO \XYZ\dsi\parasol.bmp for OS/2

IDB_LOGO BITMAP \XYZ\dsi\parasol.bmp for Windows

to refer to any bitmap that you create.

For OS/2, the bitmap must be an OS/2 bitmap. For Windows it must be a Windows bitmap. In order for you to use Microsoft Windows bitmaps in OS/2, Parasol Developer includes a bitmap conversion utility. This utility will convert MicroSoft Windows bitmaps to OS/2 bitmaps. The bitmap conversion program can be found in the Parasol Developer directory \EXE and is named BMP.EXE. This program is not required for the Windows version, and therefore is not included.

To run the bitmap conversion program, change the directory to where you find the BMP.EXE program and type:

BMP {input filename} {output filename}

An example would be:

BMP C:\WINDOWS\CHESS.BMP D:\XYZ\DSI\MYCHESS.BMP

When you run this program, a simple OS/2 window will appear with just the File menu. This menu provides the various options for converting bitmaps between various versions of Windows and OS/2. Select the option and the conversion will operate on the files you named above.

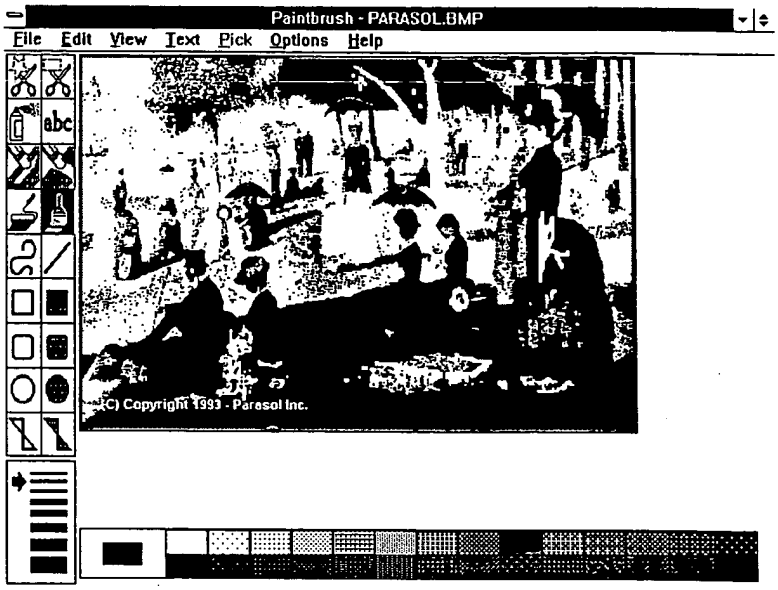

Figure 7-2: Using Painbrush in Windows to edit the standard Parasol Bitmap.

For either OS/2 or Windows, you can change the picture that appears when your users start the application you are building for them.

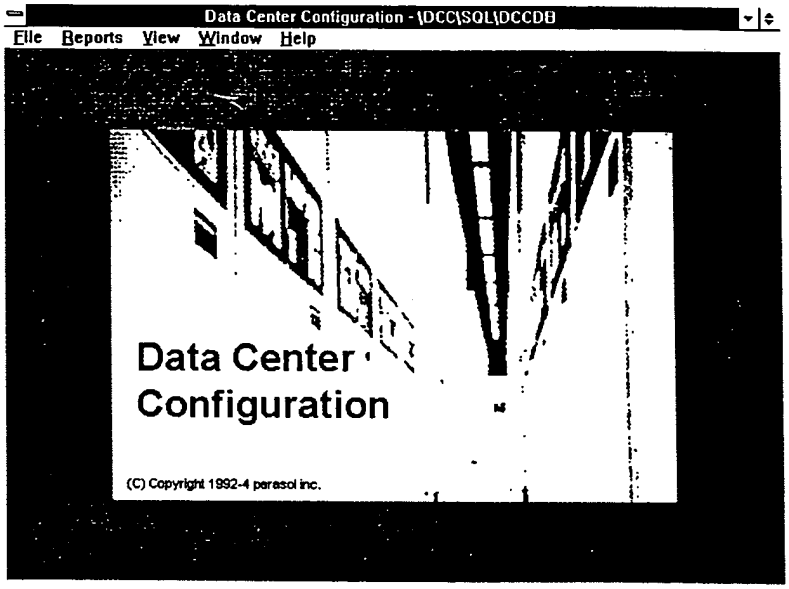

Figure 7-3: Example startup screen with customized bitmap.

7.3.2 Insert Application Icon

In the \DSI directory you will find an OS/2 or Windows icon called PAR.ICO. This is the default application icon. You can change the default icon (which is a parasol) by editing the file in your \DSR directory named XYZMAIN.RC where xyz is the name of your project.

Change the line

POINTER IDW_MAIN \XYZ\dsi\par.ico for OS/2

IDW_MAIN ICON \XYZ\dsi\par.ico for Windows

to refer to any icon you create. Icons are created using the Icon Editor that comes with the IBM OS/2 Software Development Kit or the Microsoft Windows Software Development Kit..

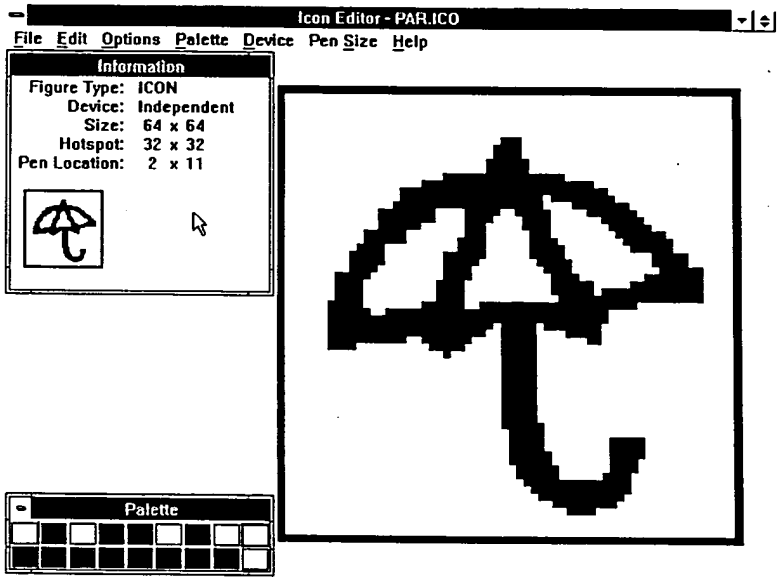

Figure 7-4: Enlarged view of the Parasol standard icon in the OS/2 Icon Editor.

By using this technique, you can change the icon image that will appear in the Parasol Group or Folder window for your application when your users install it on their system.

7.3.3 Change Accelerator Table

In the \DSR directory you will find a file named XYZACLR.RC. This file contains a standard OS/2 or Windows accelerator table. To modify this file you will need to refer to the section on accerator tables in the IBM OS/2 Software Development Kit, the Microsoft Windows Software Development Kit, or contact Parasol Technical support. If you add an accelerator key to the table remember to add the key to the application menu file.

For example, lets say we have a menu item called IDM_MY_MENU_ITEM_ID and we want to run this menu item from the keyboard using an accelerator key like control INS on the keypad.

Insert into the accelerator file:

OS/2: **VK_INSERT, IDM_MY_MENU_ITEM_ID, VIRTUALKEY, CONTRO**
Windows: **VK_INSERT, IDM_MY_MENU_ITEM_ID, CONTROL, VIRTKEY**

Insert into the menu file for this menu item:

OS/2: **MENUITEM "~My Action\t^INS", IDM_MY_MENU_ITEM_ID, MIS**
Windows: **MENUITEM "&My Action\t^INS", IDM_MY_MENU_ITEM**

The "\t^INS" is interpreted as TAB and the string ^INS to signify the control INS key on the keypad. You can use these functions to change the accelerator keys currently in the Parasol Executive default behavior, or to add new ones.

7.4 Compile and Link the Application

This procedure is where the majority of work is done by your computer system in compiling all your objects, linking them together with the Parasol Executive libraries, and making an executable application.

When you created the project with the Parasol Source Code Generator it created a directory \DSO. The \DSO directory holds the application make files to compile and link an application. These files are written by the Source Code Generator.

7.4.1 Development and Production Versions

A Parasol application can be compiled and linked in two ways. A development compile and link, or a production compile and link.

The Parasol Executive libraries, and your own executable program, will be twice the size if compiled and linked for development purposes rather than for production purposes. This is because nearly half the executable code performs error checking while in development mode. A Parasol application in development also runs approximately 8 times slower than a production version. Always develop using development libraries. When you are ready for beta testing, switch to compiling and linking with production libraries. How to do this is discussed later in this chapter.

When you installed Parasol Developer the following directories were created.

- \DLLD - Parasol Development Run-time Dynamic Link Libraries
- \DLL - Parasol Production Run-time Dynamic Link Libraries
- \LIBD - Parasol Development Link libraries for linking your application.
- \LIBP - Parasol Production Link libraries for linking your application.

When you are finished with developing your application, you can remove the \DLLD subdirectory from your LIBPATH statement in your CONFIG.SYS for OS/2, or the path in the AUTOEXEC.BAT file for Windows.

7.4.2 Prepare Link File

The file XYZ.LNK in the DSO directory, where xyz is the name of the project, is the application link file. It contains statements that link the compiled objects that you have created in a later step with the Parasol link libraries.

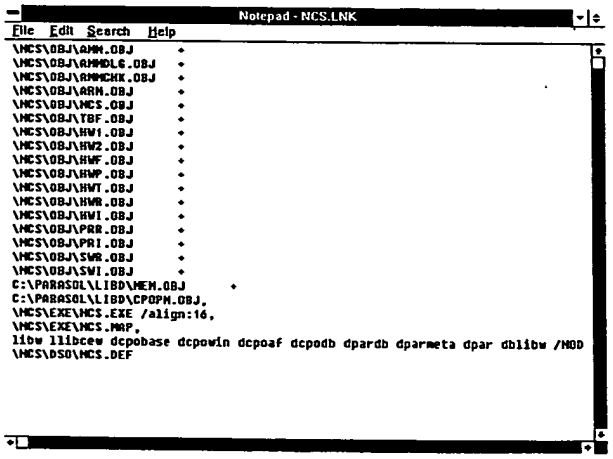

Figure 7-5: Example listing of the NCS.LNK file.

Run the Source Code Generator and select Utilities - Link File. The link file and the setup file are the only files that are affected by the difference between production and development. Therefore this is an option on the menu.

In this file you should notice that it refers to the directory \LIBD. This is the location of the development libraries. When you are ready to switch to production, this will change to reflect the production libraries in the directory \LIBP.

Also note that the file SETUP.CMD for OS/2 or SETUP.BAT for Windows (see section 7.2) that runs your project environment has the statement:

```
set LIB=c:\C700\lib;c:\parasol\libd;c:\xyz\exe;c:\databaselib;
```

where databaselib refers to the database libraries for the version of Parasol you are using. You should also change the reference in this file from LIBD to LIBP when you move into production mode. This is the location that the linker uses to search for libraries to link to or you can use the Source Code Generator to rewrite the file.

7.4.3 Prepare Make File

The file XYZ.MAK in the DSO directory, where xyz is the name of the project, is the application make file. It contains statements that compile the objects that you have created using the Parasol Source Code Generator. The format of the make file is MicroSoft or IBM Make Utility syntax. This is the standard make syntax that comes with both C compilers.

When you run the Parasol Source Code Generator, it writes the classes (objects) into the form described in Appendix B. In other words, four files are created per class. These classes are compiled using the MicroSoft C Compiler or the IBM C SET/2 Compiler, whichever you are using. Each class that the generator has written must be compiled.

The make also runs two other compilers.

- The Resource Compiler that comes with the IBM OS/2 Software Development Kit or the Microsoft Software Development Kit.
- The SQL Preprocessor that comes with the Database Manager, XDB, or WatcomSql.

Run the Source Code Generator and select Utilities - Make File.

There are three major areas in the make file.

- The to do list for all C objects at the top of the file.
- A multi-line entry for each SQL Interface Object in the application to run through the SQL Preprocessor (Note that SqlBase does not have this compiler).

For Database Manager, it is five lines that look like this:

```
$(SDIR)\.....
   startdbm
   sqlprep $(SDIR)\$(@B).sqc XYZDB etc.....
   sqlbind $(BDIR)\??i.bnd XYZDB
   stopdbm
```

For XDB, it is two lines that look like this:

```
$(SDIR)\....
   PRECOMP $(SDIR)$(@B).sqc /m 20 /l etc.....
```

For WatcomSql, it is two lines that look like this:

```
$(SDIR)\....
   sqlpp -oWINDOWS $(SDIR)\$(@B).sqc $(SDIR)\$(@B).c
```

- One 2 line entry for each C Object in the application to run through the C compiler.

```
$(ODIR)\??1.obj: $(SDIR)\??1.c $(HDIR)\??1.h $(HDIR)\??1cls.h

cl >$(ERRS) -Fo$(ODIR)\  $(MEM)  $(CN) $(SDIR)\$(@B).c -c
```

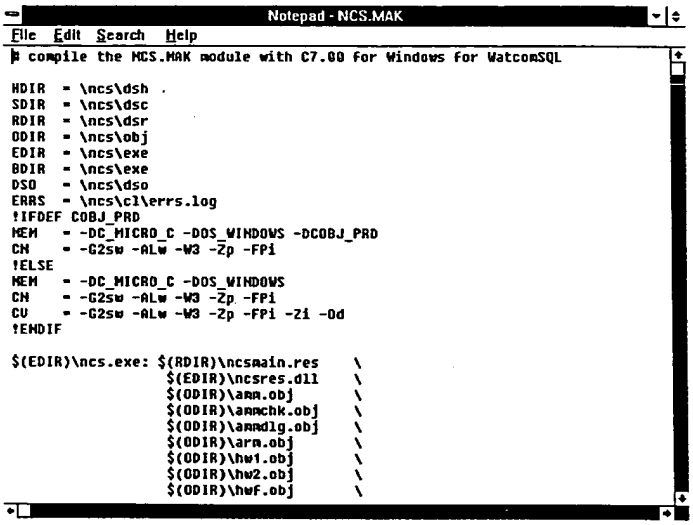

Figure 7-7: Example listing from the make file for the NCS application.

7.4.4 Notes on Compiling

Parasol libraries are compiled for the in-line floating point math package. Your code must be compiled for the same package. This is how the make is organized.

For OS/2 only, Parasol libraries are compiled for multi-threading. Your code must be compiled the same. This is how the make is organized.

The make file is designed to pick up where it left off when it encounters fatal error codes. You can review the log of error messages (see below), fix the problems, and then start the make file again to continue from the last location where successful compilation took place.

7.4.5 Command Line to Run Make Application

When you created the project with the Parasol Source Code Generator it created a directory \CMD. The \CMD directory holds the application command files. In this directory are two files:

M.CMD for OS/2 and M.BAT for Windows - If you type this command the make utility will create a development executable.

The command statement is: **nmake -f \xyz\dso\xyz.mak**

MP.CMD for OS/2 and MP.BAT for Windows - If you type this command the make utility will create a production executable.

The command statement is: **nmake COBJ_PRD=TRUE -f \xyz\dso\xyz.mak**

The command line argument for the make COBJ_PRD tells all the C objects to compile themselves for production. When you are ready, type either M or MP at the system prompt in the Project full screen window for the project application you are working with. Also, remember to change your SETUP.CMD or SETUP.BAT file to reflect the correct libraries and include files or you can use the Source Code Generator to rewrite the file..

7.4.6 Error Log File

When you created the project with the Parasol Source Code Generator it created a directory \CL. The \CL directory holds the error log output from the make procedure which is named ERRS.LOG. You should always check the log file for warning messages (errors will stop the make).

If the design specification in Parasol Designer and the Parasol Source Code Generator is correct, you should only get the following warnings on SQL Interface Objects when working with Database Manager or XDB:

- warning C4047: '=' : different levels of indirection
- warning C4047: 'argument' : different levels of indirection
- warning C4024: 'strlen' : different types : parameter 1
- warning C4047: 'argument' : different levels of indirection
- warning C4024: 'sqlasets' : different types : parameter 2

These warnings can be ignored and is caused by a syntax requirement in the SQL Preprocessor for those databases.

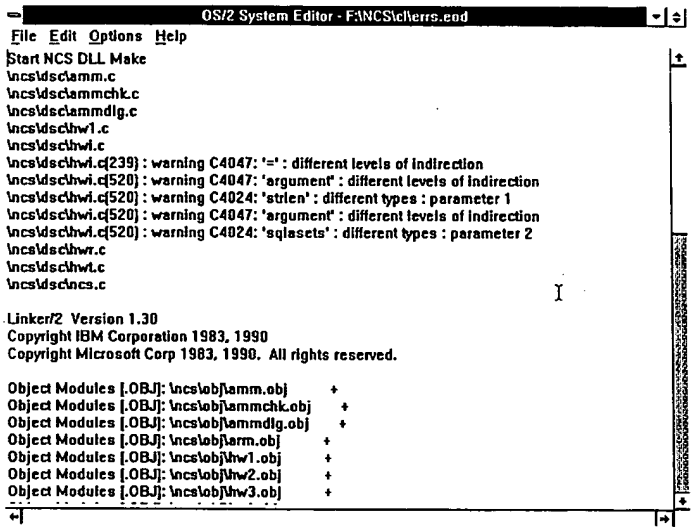

Figure 7-7: Sample listing of the ERRS.LOG file from compiling the NCS application.

The most common mistakes are to declare a database column as say, INTEGER and declare it in the Parasol Designer as MEDIUMINT. These types are not compatable since MEDIUMINT takes two bytes of memory and INTEGER in the database is a 4 byte number. Other examples are real numbers equated with a date and so on. The compiler will not catch database text as dates in the user interface. Your user will spot it quickly however.

Again, you should always check the log file for warning messages. A little warning can cause the application to generate a protection violation. This means it will blow up.

7.4.7 Library Organization

The Parasol Developer comes with precompiled object code organized into dynamic link libraries and their associated library files. Libraries are provided for creating debugging and production executables. This allows you to test your code with the larger, slower debug libraries which help you catch bugs before they cause a problem. Then when you are ready to create a production executable, you can link with the production libraries which are much faster and smaller, with all the debugging code removed. The following list describes each library provided.

Production Libraries:

- CPOBASE.LIB C+O Class Library Volume 1
- CPOPM.LIB C+O Class Library Volume 2 for OS/2
- CPOWIN.LIB C+O Class LIbrary Volume 2 for Windows
- CPOAF.LIB C+O Class Library Application Frameworks
- CPODB.LIB C+O Class Library Database Functions
- CPODBM.LIB C+O Class Library Database Manager
- CPOGUP.LIB C+O Class Library SqlBase
- CPOWAT.LIB C+O Class Library WatcomSql
- CPOXDB.LIB C+O Class Library XDB

- PAR.LIB Parasol Floating Point code
- PARMETA.LIB Parasol Kernal code
- PARDB.LIB Parasol Database I/O code
- PARDBM.LIB Parasol Database Manager system tables
- PARGUP.LIB Parasol SqlBase system tables
- PARWAT.LIB Parasol WatcomSql system tables
- PARXDB.LIB Parasol XDB system tables

Note that the development libraries have the same names, but with the letter D in front of each name. The library PAR.LIB contains the floating point code for Parasol Executive and can not reside in a dynamic link library.

7.5 Load Application Specific Data

At this point you are ready to insert data into the Parasol system tables that you have used. See Appendix A for details on the system tables.

7.5.1 What to do with the System Tables

- PAR_ABSTRACT — Nothing
- PAR_PROCEDURE — Nothing
- PAR_CONNECTION — Nothing
- PAR_ITEMTYPE — Nothing
- PAR_ITEM — Nothing
- PAR_SELECTION — Use the query functions of your database and insert the selection values you want for:

  Type AST (Abstract type selection)
  Type BPT (Procedure type selection)
  Type RVR (Revision reason selection)
  Type RVS (Revision severity selection)

and any other combobox or listbox lists you have added to the application. The type BRD (Bitmap type selection) is reserved and should not be modified.

- PAR_BITMAP — Use the query functions of your database and insert references to any pictures required by the application.
- PAR_REVISION — Nothing
- PAR_PARMSG — Nothing
- PAR_PARCONTROL — Nothing
- PAR_HELPREF — Nothing
- PAR_DIALOG — Nothing
- PAR_DLGCONTROL — Nothing

Chapter 8 - Finishing Touches

8.1 Introduction

After you have successfully compiled and linked your application, you can enhance it with several finishing touches covered in this chapter. These include:

- user and workstation configuration options
- context sensitive help for your application
- place dialog boxes into a dynamic link library
- create an installation procedure for users of your application

These are described in further detail below.

8.2 Configure the Application by User and Server

You can configure the application by user and server for colors, fonts, layouts, what works for what people, choice lists, additional security (above what the database provides), filters, startup look an feel and much more.

This is achieved by giving each user a private profile file combined with data in the system table PAR_ITEMTYPE. The PAR_ITEMTYPE table contains card design data. All other data is stored in the private profile file named PAR.INI. Parasol reads the private profile file from the working directory that you run the Parasol application. Note that in OS/2 the PAR.INI file is not editable because it has imbedded control characters for Parasol to read in interpret.

You can create two versions of a Parasol application. An operator version and an administrator version. The administrator version can have access to all the configuration menus in Parasol, and the operator version will not. The operator version would be installed with the profile file as part of the installation process. Thereby, the operator will have a pre-configured version of the application.

8.3 Add Context Sensitive Help to Dialog Panels

When you create dialog panels, you can place a button on the panel and name it DID_HELP. When the user pushes this button, Parasol Executive will query the

PAR_HELPREF table, of the Parasol system tables, and retrieve one record based on the dialog box id you assigned to the dialog the user was in when they pushed the Help button.

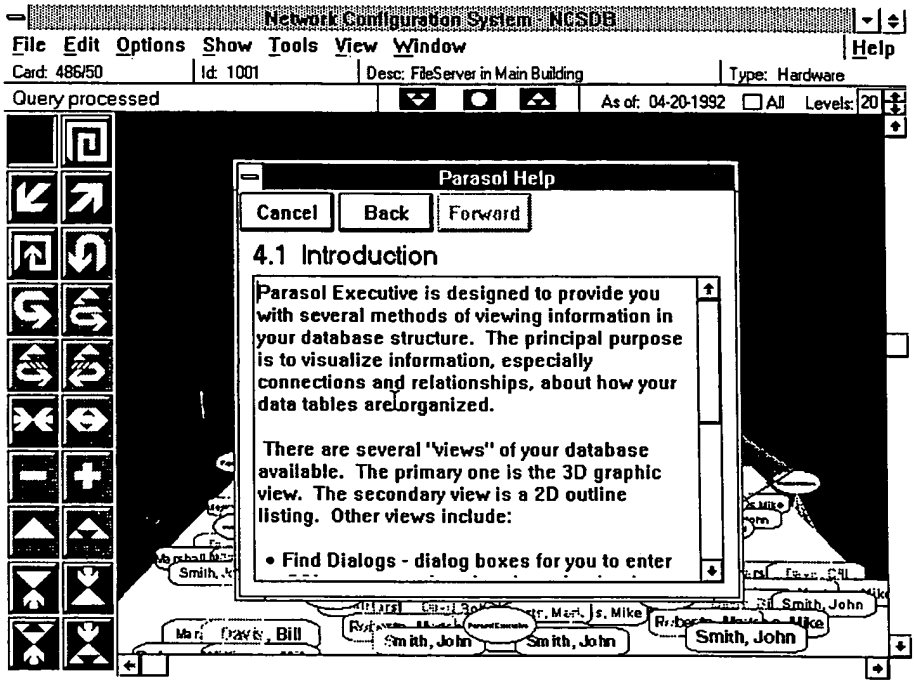

Figure 8-1: Sample help screen from the NCS application.

This record contains:

- the help file name
- the key that is the entry point to your text in the help file

The key in the PAR_HELPREF table is the actual number of the dialog box. You must get this number from your resource definition file, XYZDEF.RH, where XYZ the the name of your project. The actual number is required because all your names of objects are converted to the numbers in your #DEFINE statments when the application is compiled.

Included with Parasol Developer is a Help compiler that reads a plain ASCII text file and outputs it to a Parasol Help File.

Parasol help can consist of :

- table of contents sections
- body text sections

Each line in the table of contents section can run another section in the file. Body text is the bottom of the tree, and the user can only go back to the Table of Contents section that called it. This help is the same help format that comes with all the Parasol products.

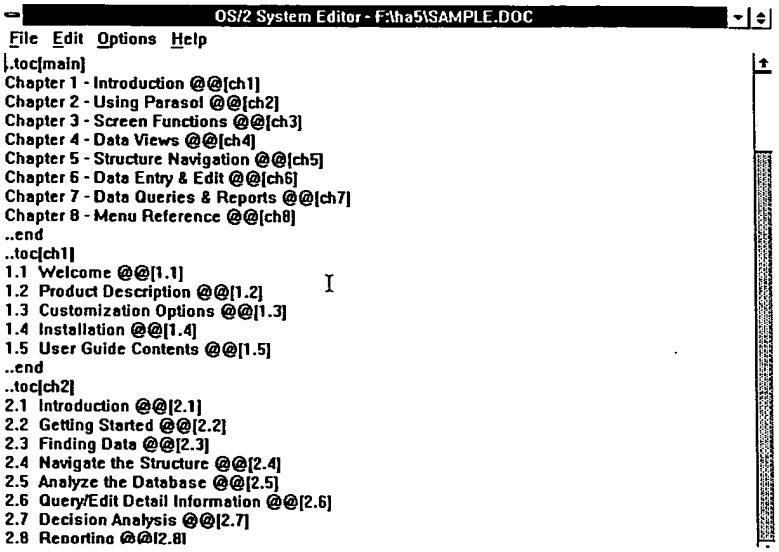

Figure 8-2: Sample Table of Contents help file.

The help file you create uses plain text combined with a markup language. The syntax is simple as shown in the following example:

Table of Contents Section:

```
..TOC[MAIN]
                Chapter 1 - Introduction @@[ch1]
                Chapter 2 - Using Parasol @@[ch2]
                Chapter 3 - Screen Functions @@[ch3]
                Chapter 4 - Data Views @@[ch4]
..END
```

The "..TOC" and "..END" statements signify the start and end of the section, where "MAIN" is the key name of the section. Your top level Table of Contents section MUST be named "MAIN" because Parasol Executive will go to the key "MAIN" if it cannot find the text section that it is looking for. The "@@[ABC]" statement is the section to run if the line is selected, where "ABC" the key name. Only one line per command is allowed.

Body Text Section:

..BODY[2.1]
This chapter describes the basic operations of Parasol Help as an introduction to how the system works. Further details about the window areas, navigation etc., etc....
..END

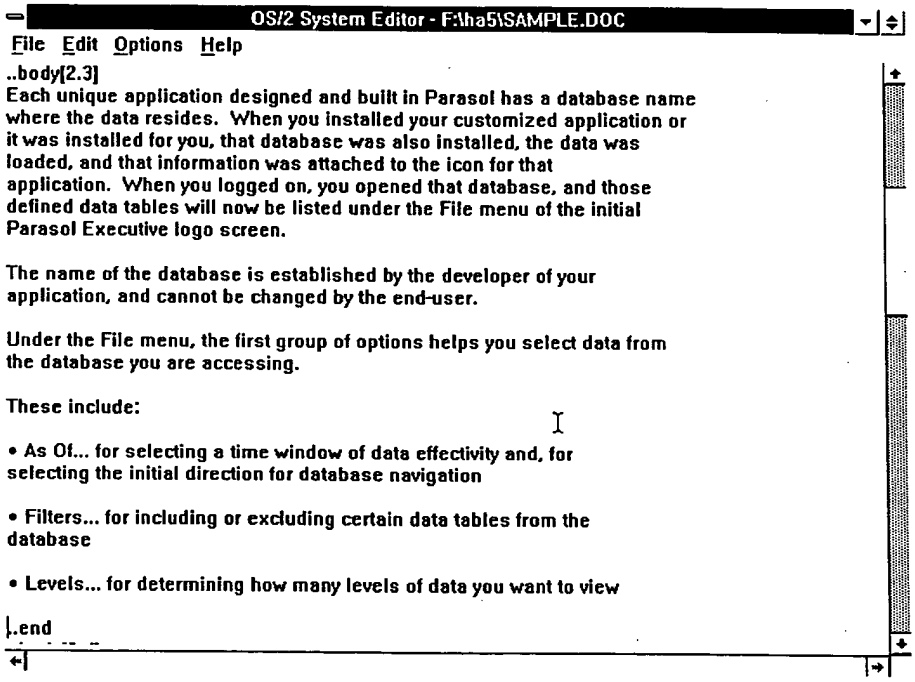

Figure 8-3: Sample body text of a help file.

The "..BODY[EFG]" and "..END" statements signify the start and end of the section. Any text placed between these statements becomes the text the user sees. The characters "EFG" are the key name of the text that follows.

Formatting issues:

Parasol Help displays justified paragraphs. Paragraphs are defined as breaking on the first blank line.

To run the Parasol Help compiler first create your help text file using your favorite text editor. This file would normally be the output from a desktop publisher or a word processor. Markup the file and run the compiler by double clicking on the Help Compiler icon in the Parasol Group.

Select the Set Compile Parameters option of the File menu and fill in the appropriate boxes in the dialog that appears.

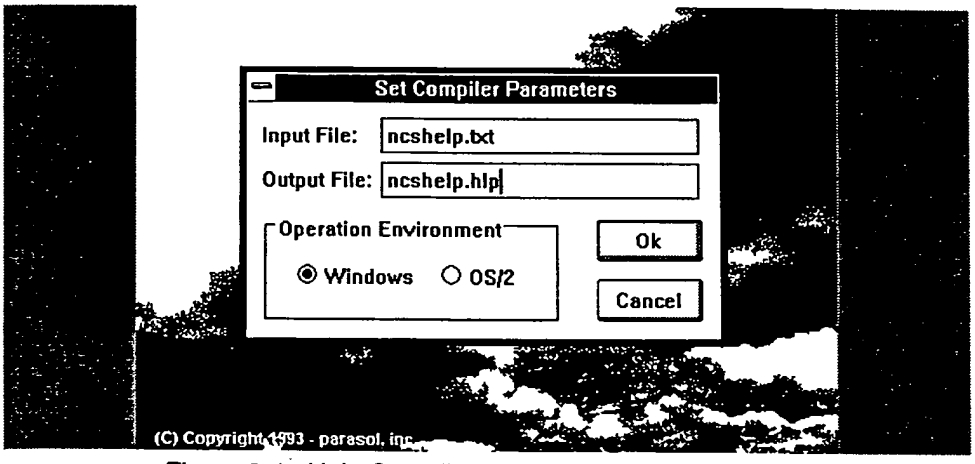

Figure 8-4: Help Compiler Dialog box with parameters.

Input file: This is the help file your have created

Output file: This is the file that Parasol will read to display the help.

Environment: Select whether you are using OS/2 or Windows

The Parasol Help compiler is able to report if you use keys that are not found. Comments cannot be inserted into the file. You may insert any valid codepage character to create bullits and other symbols. Codepages are IBM's language specific character sets and are detailed in the IBM OS/2 Software Development Kit or Microsoft Windows Software Development Kit for the Windows specific ANSI character set.

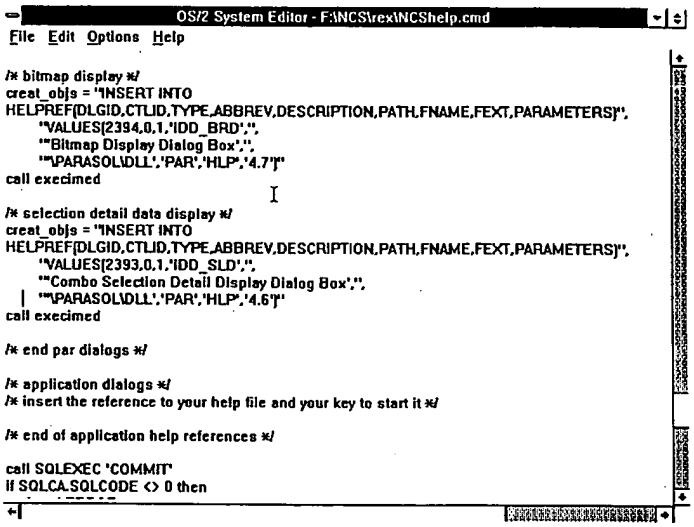

Figure 8-5: Sample listing of the NCSHELP.CMD file for the NCS application.

Create a record in the PAR_HELPREF table that corresponds with each dialog help button that you have in the application. The file named XYZHELP.CMD in the \REX directory of Parasol Developer shows many examples.

These records are Parasol Executive's own dialogs. The disk drive name is optional and only used if the help file is not found on the same disk that Parasol Executive is running from.

Sample PAR_HELPREF record insert statement:

```
INSERT INTO
HELPREF(DLGID,CTLID,TYPE,ABBREV,DESCRIPTION,PATH,FNAME,
FEXT,PARAMETERS)

    VALUES(2312,0,1,'IDD_BPL',

    'My Dialog Box',

    '\PARASOL\EXE','XYZ','HLP','4.5';
```

8.4 Place all Dialogs into a Dynamic Link Library

When an application is to be put into production, the dialog panels can be stored in the application resource Dynamic Link Library. This is done to increase performance of an application, especially if the database is on a server. This is the last step before the application goes into production, and is configured with an installation procedure. This is not to say that you cannot install a development program with the installation procedure in the next section, but it will not be final.

Dialog panels can be saved to disk or database in Parasol Designer. The PAR_PARCONTROL record of the database is used by Parasol Executive to determine if dialogs are to be read from the database or an application resource Dynamic Link Library. By default the dialogs and their controls are read from the database.

Additional fields are stored in the database to describe the behavior of each dialog, and are used by the Parasol Source Code Generator. These fields can not be stored in the disk version of the files. Therefore, when you are developing the application and running the Parasol Source Code Generator, the dialogs and controls are saved to the database.

Finally, when all the dialogs are cast in stone, convert them to disk files and save them in the \DSR directory. As discussed in earlier sections, this is good practice all the way through your development process. Parasol Designer includes a menu ption to write all dialogs to disk.

Once the files are in .DLG file form they can be compiled with the Resource Compiler that comes with the IBM OS/2 Software Development Kit or the Microsoft Windows Software Development Kit. To do this, edit the file in the \DSR directory named XYZDLL.RC, where xyz is the name of the project. Insert the line,

rcinclude \XYZ\dsr\???.dlg

for each dialog in your application. Table View dialogs should NOT be included here, they do not need a resource definition.

The dialogs should always be named the same as your "C" object that uses the dialog.

Now change the PAR_PARCONTROL record in the database using the SQL query functions of your database The column to change is USEDLGDLL. This is the switch that tells Parasol Executive where to look for the dialog control information.

To use the Dynamic Link Library, set it to Y for yes. Then remake the application resource dynamic link library.

All your dialogs will now come from the XYZ.DLL file, where xyz is the project name, instead of from the database tables PAR_DIALOG and PAR_DLGCONTROL. Therefore when you install an application, you do not need to import the dialog panel specification into the database.

8.5 Create an Installation Procedure for the Application

When you created the project with the Parasol Source Code Generator it created a directory \DSO. In this directory you will find a file named INSTALL.INS. This is a file that the Parasol Installation Program uses to load Parasol Executive from floppy disks.

A basic Parasol application will consist of the following files:

- XYZ.EXE Application Parasol Executable
- XYZ.DLL Application Parasol Resource Dynamic Link Library
- PAR.INI Private Profile File
- *.BND These are only required for Database Manager. All project bind files that are created in the \EXE directory of the project
- *.abc If the database is local on the workstation then these procedural SQL files will create and load a database

where abc is CMD for Database Manager, WTS for SqlBase and SQL for XDB and WatcomSql.

- INSTALL.INS File that tells the Parasol Installation program what application files to load from the installation disks

When editing the INSTALL.INS file, be careful not to change the statements that load the kernel Parasol software.

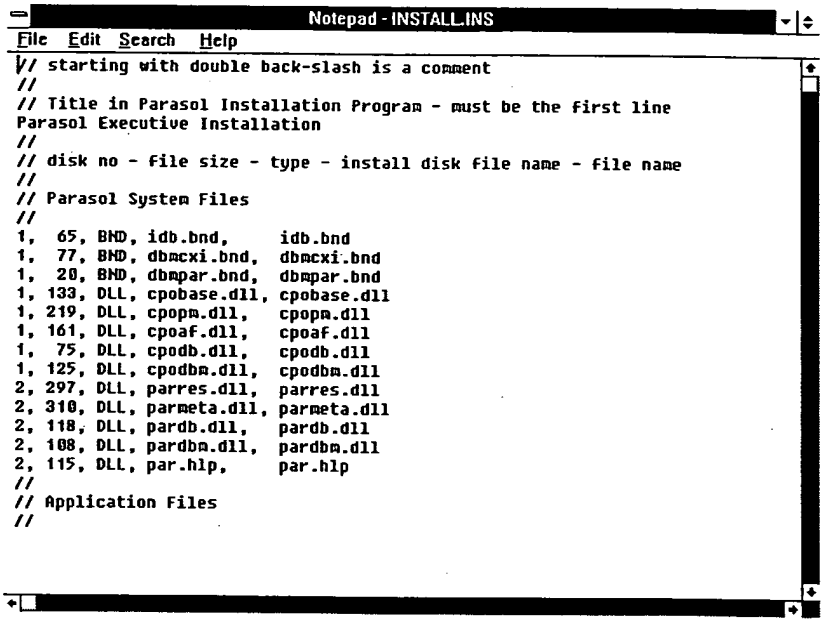

Figure 8-6: Sample listing of an INSTALL.INS file.

The project INSTALL.INS file has the following format:

- Column 1 - Disk number
- Column 2 - File size in kilobytes
- Column 3 - Directory to put file in. Select:

  BND - Bind files
  DLL - Dynamic Link Libraries
  EXE - Executable file and .INI files
  REX - Procedural SQL files
  any 3 character directory name which be created by Parasol Install

- Column 4 - File name on floppy disk
- Column 5 - Name on Hard Disk

Make sure you put commas between each of the columns in the actual file.

To create the application icon for an application, additional parameters are assigned to the appropriate application EXE file. These are:

- Column 6 - Group name
- Column 7 - Title for the icon in the group
- Column 8 - Parameter 1 for the icon
- Column 9 - Parameter 2 for the icon
- Column 10 - Parameter 3 for the icon

If the group does not exist, a new group will be created. If it does exist, no new group is created.

A new icon is created and loaded whether one exists or not in the group.

An example statement for creating a new group and icon might be:

3,284,EXE,NCS.EXE,NCS.EXE,Parasol,Network Tracking,e:\sqldb\ncsdb

To automatically load a readme file for the application (e.g. special installation notes or instructions) place the following statement as the last line in the installation file:

LOADFILE, EXE, XYZREAD.ME

where EXE is the directory you want to load the file into, and XYZREAD.ME is the name of the file to load.

To set the default directory to load the application place the following statement as the last line in the installation file:

DEFAULTDIR, C:\XYZ

Place the INSTALL.INS file on the Installation disk together with the Parasol Executive files. Place your application files on the disk(s) that you require for your application. Mark the disks with a disk number so the Parasol Install Program will read it to confirm the user has placed the correct disk in the drive.

How to mark the disk(s):

Place the empty or full disk into the floppy drive.
Set your working directory to the root directory of the floppy disk.
From the OS/2 or DOS prompt type:

ECHO (C) My Copyright Notice - 1993/4 > DSK.@

where @ is the disk number

Example: ECHO (C) Parasol, Inc - 1992/4 > DSK.3

Now you are ready to duplicate the disks you have made, and distribute them to your users.

Congratulations!

You have now completed a production application generation process. Ready to start again with another one?

Appendix A - Design Database

Parasol works with tables defined in several industry standard SQL databases. Some of these tables might currently exist, or they may need to be defined. One of the advantages Parasol provides for databases that already exist is the ability to connect rows of the same or other tables without restrictions, other than cycles. This implies that structure can be retrofitted to databases whose original design did not provide for data relationships, without the need for foreign keys.

A.1 Design Tables

In addition to much literature on the subject, Parasol adds the following considerations when designing a database.

A.1.1 Connectivity

Lets take a simple example. We have two tables, the department table and the person table. Given SQL as our access to these tables and we want to determine, for a given person record, in which department the person works. In order to accomodate this sort of query, we need to place a column in the person table identifying which department the person is in, in other words, the key of the department table (FOREIGN KEY).

With Parasol this ability to link records together is achieved by graphically attaching the two records together, or by inserting the parent - child relationship into the Parasol system PAR_CONNECTION table. The PAR_CONNECTION table holds all links between rows. The PAR_ITEM table holds the keys of all connected records in the database. So if a link is added, both the parent key and the child key must exist ONCE in the PAR_ITEM table.

Therefore, if you need to link records (and you probably would not be reading this if you didn't), let PARASOL take care of these problems for you.

A.1.2 System Tables

In order to design a database we must understand the workings of Parasol, the system table layouts, and what they are for.

Parasol contains the following system tables:

PAR_ABSTRACT - a sample table that comes with a ready made user interface. This table can be used as Parasol comes out of the box. You can use this table to do prototyping and building representative sets of data. The abstract table contains a TYPE column that enables a prototyper to filter records based on the type. This table can also be left in a final application to allow connected data entry into a database that has no specific table for it to go.

If you do not want to use this table delete it from the menu. The supplied name for this table in the menu is "Sample".

The PAR_ABSTRACT table comes with a PAR_BITMAP column. A PAR_BITMAP column can be added to any table you design. The column contains a key in the BITMAP table thereby allowing a single picture to be attached to multiple records. A bitmap column can be named anything you like, but must be defined as CHAR(10) in the table creation statement.

PAR_PROCEDURE - This is a system table that comes with a ready made user interface. This table allows the application designer to attach procedures to records. Since you might want many procedures for a given record, these are implemented as any other Parasol user table, and can be linked. This record contains DDE protocalls, including a disk file location specification. This enables Parasol to be attached to other applications in OS/2 or Windows, and communicate with them to load data. If the application you want to start does not support DDE, it can at least be started with command line arguments derived from the PAR_PROCEDURE record.

PAR_CONNECTION - This table is what gives Parasol the ability to relate (link) rows in tables. The table has the following columns:

- PARENTTYPE - Parent record table type. This is the number defined in the ITEMTYPE table TYPE field.
- PARENTID - Parent record key
- COMPONENTTYPE - Child record table type. This is the number defined in the ITEMTYPE table TYPE field.
- COMPONENTID - Child record key
- FINDNBR - {Reserved for future use}
- REVLEVELIN - Revision level of the in effective connection

- TYPE - Type of connection
- EFFECTDATEIN- Date the connection is in effective
- ENGORDNBRIN - Change order that created the connection
- REVLEVELOUT - Revision level of the out effective connection
- EFFECTDATEOUT - Date the connection is out effective
- ENGORDNBROUT - Change order that out revised the connection
- SYSDATE - Date the record was created/modified

PAR_ITEMTYPE - This table contains the definition of tables that you are going to add to Parasol. When adding a table that Parasol should be aware of, an entry MUST be made in the PAR_ITEMTYPE table. You add new entries and edit this table in the Parasol Source Code Generator.

The columns that must be filled in are:

- TYPE - Number that is the table id. You can use the numbers 100 through 32000 for this id. This value is also inserted to the "C" object definition file named LIBXYZ.H. See section 3.4.2
- CODE - Name that the end-user sees for this table in Parasol.
- TABLENAME - Name of table as defined in the SQL create table command.
- DESCRIPTION - Description of table. Not displayed in Parasol.
- CARDDISPLAY - Column to be used for displaying items in Parasol.
- SUBTYPE - Type of PAR_ITEMTYPE record - TABLE and DESIGN are the two possible values.

There are over 20 other columns in this table. These define the appearance of this table's records in Parasol's outline and 3D views. Basically, these include color, shape and font definitions. None of this data needs to be filled in, since Parasol will use defaults. Or a user can customize these table columns, which are modified in Parasol's menu Options Design Cards and Options Design Outline.

Records can be added to this table and given a subtype value of "DESIGN". These records will appear in Parasol's Options Design Cards and Options Design Outline. These designs can be used in the object TBF (see section 3.4.1) to add limitless card and outline design possiblities.

PAR_ITEM - This table contains the table type and key of all connected data that should be examined when Parasol goes through its implosion/explosion algorithm. When entering connections through Parasol, this table is automatically maintained. If data is loaded directly into the database, the item records must be inserted as well -- once for each row in a table.

The following columns are used in this table:

- TYPE - Number that is the table id. You can use the numbers 100 through 32000 for this id. This value is also inserted to the "C" object definition file named LIBXYZ.H.
- ID - Unique ID of the element in a network.
- CARD - If this column is not NULL than Parasol will override the card display with this string.
- SYSTEMDATE - The date the record was created.

PAR_SELECTION - Whenever a combobox or listbox appears in a Parasol application, the user is given a list of items to pick from. We can define a selection type for any combobox or listbox that is used on a dialog box. When the combobox or lisbox is made visible, it loads the data the user sees from the ID column that matches a given TYPE column. Therefore an application designer can create choice lists of any data element. The remaining fields, DESCRIPTION, USER1, USER2, USER3, USER4, and NOTES are additional data fields the application designer may wish to associate with the choice. For example USER1 might be the field that is inserted into your table based on a selection.

The following are the columns in this table:

- TYPE - Type of selection
- ID - What the user sees
- DESCRIPTION - Description field
- USER1 - String buffer

- USER2 - String buffer
- USER3 - String buffer
- USER4 - String buffer
- NOTES - Notes attached to selection

By double clicking the combobox arrow (not supported on listboxes), a user can access the details of any of the selection record data through a dialog box. The data may be changed in this dialog, but not created. Use SQL to add data to the table.

The table contains some sample data for the following system comboboxes:

- Type AST - Abstract type selection: NETWORK, ORGCHART, OBJECT
- Type BPT - Procedure type selection: DIAGNOSE, EXCALATE, EXPLAIN, REPAIR, SERVICE
- Type RVR - Revision reason selection: CLOSE, CORRECT, INSTALL, MOVE
- Type RVS - Revision severity selection: RECORD, FIX
- Type BRD - Bitmap type selection: MSWINDOWS, OS/2

PAR_BITMAP - This table is a list of bitmap id's and there location on the disk. You can declare a combobox or listbox to be of the PAR_BITMAP table type instead of the PAR_SELECTION table type referred to above. The combo box or listbox loads the list of bitmap id's found in the PAR_BITMAP table.

- ID - Id of the row in the table
- TYPE - Type of bitmap (1 = MSWINDOWS .bmp file or 2 = OS/2 .bmp file)
- DESCRIPTION - Description attached to file
- DRIVE - File drive
- PATH - File path
- FNAME - File name

- FEXT - File extension

PAR_REVISION - When connections are added in Parasol, or when they are disconnected (not discarded), a revision record is added to the PAR_REVISION table for the parent record. Hence a list of revisions to connections is available for any record in the database.

The revision record contains the following columns:

- TYPE - Record table type. This is the number defined in the PAR_ITEMTYPE table TYPE field.
- REVID - Key of parent record.
- REVLEVEL - Revision levels. This is automatically incremented as changes are made to the connections of the parent record.
- EFFECTDATE - The date the connection is effective
- ENGORDNBR - Change order associated with the connection.
- REASON - Reason, look up in PAR_SELECTION table.
- SEVERITY - Severity, look up in PAR_SELECTION table.
- IMPLEMENTDATE - Implementation date of connection.
- APPROVALDATE - Approval date of connection, if any.
- NOTICEDATE - Notice date for connection. For reporting changes.
- SYSDATE - Date the record was created/modified
- NOTES - Note that is attached to revision.

PAR_PARMSG - When connections are made in effective or outeffective a record is added to this message table indicating the record, date and a message id.

- ID The value 105 (other values are planned)
- MSGDATE - Date the record was created
- MSGDATA - ID if record being created/modified

PAR_PARCONTROL - This table contains records that tell Parasol how to behave at runtime. Refer to section 8.2.

The following columns are used in this table:

- ID - Unique ID
- USEDLGDLL - Use dialog resources from resource DLL (Y/N)
- DATEFORMAT - Date format to be used to communicate with database: a value of 1 gives MM DD YYYY, a value of 2 gives DD MM YYYY, and a value of 3 gives YYYY MM DD.
- TIMEFORMAT - Time format to be used to communicate with database: a value of 1 gives HH MM, a value of 2 gives HH MM SS (24 hour clock).

PAR_HELPREF- Lookup table for the on-line help system. It contains the following fields:

- DLGID - Dialog box id from the zxyDEF.RH definitions file where xyz is the name of your project.
- PARAMETERS - Parameters for running help
- TYPE - Type of file
- ABBREV - Help abbreviation
- DESCRIPTION - Description attached to help reference
- DRIVE- File drive
- PATH - File path
- FNAME- File name
- FEXT - File extension
- CTLID - {reserved for future use}
- OPERATOR - {reserved for future use}

PAR_DIALOG - Table containing the definition of dialog boxes defined by the application developer. Dialog boxes are edited using the Parasol Designer.

- A data entry dialog box that contains columns in a table.
- A "find" dialog box that creates SQL "where" clauses and allows a user to do searches on a table.
- A table view dialog box that contains one, and only one, control that is of type Table List. No buttons are required on this dialog since Parasol automatically adds a menu to the panel.
- A "report" dialog box that runs a report writer report definition and passes variables to it.

PAR_DLGCONTROL - Table containing the definition of the controls that are contained within each dialog box. Dialog boxes are designed and edited using the Parasol Designer. Graphical layout data is stored together with information needed by the Parasol Source Code Generator.

Note: After application is finalized, the PAR_DIALOG and PAR_DLGCONTROL records can be compiled into a Dynamic Link Library for improved performance.

PAR_DESIGNTABLE - Table containing the definition of a registered table. Table registrations are edited using Parasol Designer.

Information needed by the Parasol Source Code Generator is stored in this table.

A.2 Bulk Load Application Data into Tables

Data can be created interactively when the Parasol application is complete, or the data can be bulk loaded in preparation for the Parasol application when it is ready.

Depending on which SQL database product you are using, they provide a variety of ways to import data into tables including Comma Delimited files (DBase), Lotus and Excel Worksheet files, non-delimited ASCII files, and DB2 transfer files.

The PAR_CONNECTION table holds all links between records. The PAR_ITEM table holds the keys of all the connected records in the database or, put differently, all the nodes in a network. This data can be imported as with the application tables. Most applications can derive most of the connection data by parsing disk files, and the methods vary from one application to another.

Appendix B - Software Methodology

Parasol applications that are written by the Parasol Source Code Generator are written in standard ANSI "C". However, it is possible to write portions of a Parasol application in other languages and with other programming methodologies.

This appendix addresses only the Parasol Developer "C" software coding methodology.

B.1 Source Code Organization

Parasol Developer is provided entirely of C code, namely .C and .H files. Parasol source code is divided into classes (objects). Each class (object) has a long name and a three letter abbreviation. The abbreviation, or prefix, is used in naming functions, naming variables, and naming source code files. Parasol source code has the following naming conventions and associated meanings. This shows the use for a hypothetical class with prefix Xyz.

- xyz.h - This file holds the structure declaration for the class whose abbreviation is Xyz.
- xyzmac.h - This file holds the function declarations and macros for the class Xyz. This file is included if you need to perform functions against an object of class Xyz. It automatically includes xyz.h
- xyzcls.h - This file holds any function declarations and macros which are private to the class.
- xyz.c - This file holds the public function definitions for Xyz. Each function in this file is prefixed with Xyz and takes a pointer to Xyz as its firts parameter.
- SQL Interface Database Objects are a special case. They have no structure declaration.

B.2 Anatomy of Source Code

The "C" code that the Parasol Source Code Generator produces includes the header files "cobjects.h" and "cobjsys.h". The file "cobjects.h" defines many typdefs used by Parasol Classes. The file "cobjsys.h" includes "os2.h" if you are using the OS/2 version of Parasol, or "windows.h" if you are using Windows. The file "cobjects.h" and "cobjsys.h" are included with Parasol Developer. The file "cobjsys.h" has numerous INCLUDE_ definitions that can be used that tell the file what portions, or which files, to include from the operating system files.

All fundamental datatypes used by Parasol Classes are defined as typedefs in "cobjsys.h". For example, SmallInt, MediumInt, and LargeInt are int typedefs for 8 bit, 16 bit, and 32 bit quantities respectively. Str is a char, and PSTR a pointer to char. The table in Appendix C is a list of Parasol Developer datatypes.

Next let's look at the code fragment immediately following the inclusion of cobjects.h. This code fragment is how we include an H file in Parasol objects.

```
#ifndef  GDB
#include "gdbmac.h"
#endif
```

After this point the "C" file contains all the function for the object. All "C" objects have the methods CREATE, INIT, DEINIT, and DESTROY. The CREATE method creates an instance of the object, and the DESTROY method destroys the instance and any memory associated with it.

All methods (except CREATE) take as their first parameter a pointer to an instance of the object.

Parasol is an event driven environment. Your "C" objects, therefore, respond to messages. The OS/2 and Windows environments issue over a hundred messages. Your dialog objects get these messages filtered down to the messages required for dialog boxes to function.

Appendix C - Programming Interface

If you intend to use other dialog controls than those supported by the Parasol Source Code Generator, then the only class that you would be required to use to modify the behavior of a Parasol application is GDB. If you want to enhance the Parasol application beyond this, then a more detailed manual exists for understanding how to make use of Parasol Objects (Classes). Contact your local representative or Parasol Inc. to receive further information on the Parasol Programming Interface.

C.1 GDB (Generic Dialog Box Object):

The methods that exist for the GDB object are found in the file named GDBMAC.H found in the \CPO directory of Parasol developer. This file details the parameters required for each function. The first parameter, PGDB is a pointer to the instance of the dialog box. This parameter is passed to each callback that the dialog box responds to. The callbacks that are possble are found in any generated dialog box "C" object. The first parameter passed to the functions is the instance of the dialog itself and the second parameter is the instance of a generic dialog box PGDB. The methods are:

- eWrc Callback XyzInitDlg:

The dialog box is initializing itself

- eWrc Callback XyzHelp:

The help button was pushed

- eWrc Callback XyzButton1DblClk:

The button 1 double click message was received on the panel or a button. The ID of the object that received the message is passed as the ID. In this method, the second parameter passed to the function is an instance of the PWND object. Ignore this object unless you have the advanced guide.

- eWrc Callback XyzOk:

The Ok button was pushed.

- eWrc Callback XyzCancel:

The Cancel button was pushed.

- eWrc Callback XyzControl:

An object was touched, and the id of the object is passed as the third parameter to the function.

- eWrc Callback XyzCommand:

A button was pushed, and the id of the object is passed as the third parameter to the function.

The valid method calls for each Parasol Designer object are:

- Bitmap: It is recommended to use the Picture control for bitmaps displays.
- Check Box: GdbGetButtonState, GdbSetButtonState, GdbButtonClick, GdbSetButtonHilite, GdbSetButtonDefault, GdbMoveButtonHorz, GdbMoveButtonVert
- Color Palette: GdbPalSetPalette, GdbPalSetSelection, GdbPalQuery-Palette, GdbPalQuerySelection
- Horizontal Scroll: Advanced Guide
- Icon: It is recommended to use the Picture control for icon displays.
- List Box: GdbLbxSetCurrSel, GdbLbxSetNthText, GdbLbxDelNthText, GdbLbxInsertSorted, GdbLbxInsertEnd, GdbLbxClear, GdbLbxScrollPre-fixStrToTop, GdbLbxSetTopToNth, GdbLbxSetNthHandle, GdbWiden-ListBox, GdbLengthenListBox, GdbLbxSetSel
- Picture: GdbSetPcfPicture, GdbSetPcfWhiteSpace, GdbSetPcfInverted
- Radio Button: GdbGetButtonState, GdbSetButtonState, GdbButtonClick, GdbSetButtonHilite, GdbSetButtonDefault, GdbMoveButtonHorz, GdbMoveButtonVert
- Spin Button: GdbSbcGetLargeIntValue, GdbSbcGetStringValue, GdbSbcSetArrayOfData, GdbSbcSetCurrentValue, GdbSbcSetLimits, GdbSbcSetMaster, GdbSbcSetTextLimit, GdbSbcSpinDown, GdbSbcSpinUp

- Table List Box: GdbGlbGetCount, GdbGlbGetTopIndex, GdbGlbGetCurrSel, GdbGlbGetNthDpa, GdbGlbGetNthHandle, GdbGlbInsert, GdbGlbAppend, GdbGlbSetCurrSel, GdbGlbSetNthText, GdbGlbDelNthText, GdbGlbClear, GdbGlbSetTopToNth, GdbGlbSetNthHandle, GdbGlbSetSel, GdbGlbSetColumnWidths, GdbGlbSetColumnHeadings, GdbGlbGetHeadingDpa, GdbGlbGetRowDpa, GdbGlbGetColWidthDpa, GdbGlbSetHeadingFont, GdbGlbSetListboxFont, GdbGlbGetCurrButton2Sel
- Vertical Scroll: Advanced Guide

C.2 Object Names & Datatypes

The datatypes used by Parasol Developer are broken down into 5 categories.

- Core Datatypes
- Windowing Datatypes
- Database Access Datatypes
- Parasol Kernel Datatypes
- Parasol SQL Interface Database Datatypes

The remainder of this appendix lists all the datatypes by each category, including the name, definition and usage for each one.

C.2.1 Table A - Core Datatypes

Name	Definition	Usage
A3p	struct ArrayOf3DPoints	ArrayOf3DPoints class
Aop	struct ArrayOfPoints	ArrayOfPoints class
Aso	struct Association	Association class
Bis	struct BitString	BitString class
Blk	struct Block	Block class
Bool	MediumInt	Boolean Values
Char	char	Text data
Chr	char	Text data
Cls	struct Class	Class class
Dat	struct DateData	DateData class
DateFormat	enum DateFormat	Describe a display format for date
Dll	struct List	Doubly linked List class
Dpa	struct DynamicArray	DynamicArray class
Dys	struct DynamicString	DynamicString class
False	((Bool)0)	Boolean False
Fil	struct DOSFileSystem	DOSFileSystem class
Flags8	unsigned char	Bit flags 8
Flags16	unsigned short	Bit flags 16
Flags32	unsigned long	Bit flags 32
Float	float	Floating point number
Fna	struct FileName	FileName class
HashMethod	enum HashMethod	Hash methods
Hsh	struct HashTable	HashTable class
Jul	struct JulianTime	JulianTime class
LargeInt	long	Integers in range -2147483647: +2147483647:
Lel	struct ListElement	ListElement class
Lnp	struct LinePoints	LinePoints class
Mcl	struct MetaClass	MetaClass class
MediumInt	int	Integers in range -32767: +:32767
Mem	char	Memory class
Mms	struct MetaMessage	MetaSuperClass class
MODULE_NAME	_FILE_	Name of file being compiled
Msc	struct MetaSuperClass	MetaSuperClass class
Msg	struct Message	Message class
Obj	struct Object	Object class

Name	Definition	Usage
P3d	struct Point3D	Point3D class
PA3P	struct ArrayOf3DPoints *	ArrayOf3DPoints pointer
PAOP	struct ArrayOfPoints *	ArrayOfPoints pointer
PASO	struct Association *	Association pointer
PBIS	struct BitString *	BitString pointer
PBLK	struct Block *	Block pointer
PCLS	struct Class *	Class pointer
PDAT	struct DateData *	DateData pointer
PDLL	struct List *	List pointer
PDPA	struct DynamicArray *	Dynamic Array pointer
PDYS	struct DynamicString *	DynamicString pointer
PFIL	struct DOSFileSystem *	DOSFileSystem pointer
PFNA	struct FileName *	FileName pointer
PHSH	struct HashTable *	HashTable pointer
PJUL	struct JulianTime *	JulianTime pointer
PLEL	struct ListElement *	ListElement pointer
PLNP	struct LinePoints *	LinePoints pointer
PMCL	struct MetaClass *	MetaClass pointer
PMEM	Char *	Memory pointer
PMMS	struct MetaMessage *	MetaMessage pointer
PMSC	struct MetaSuperClass *	MetaSuperClass pointer
PMSG	struct Message *	Message pointer
PMTH	Void(*)(POBJ,...)	Function pointer
Pnt	struct Point	Point class
POBJ	struct Object *	Object pointer
PP3D	struct Point3D *	Point3D pointer
PPNT	struct Point *	Point pointer
PRCT	struct Rect *	Rect pointer
PSEM	struct Semaphore *	Semaphore pointer
PSET	struct SetCollection *	SetCollection pointer
PSOR	struct SortArray *	SortArray pointer
PSTK	struct Stack *	Stack pointer
PSTR	char *	Null terminated text string
PTIM	struct TimeData *	TimeData pointer
PTMS	struct TimeStampData *	TimeStampData pointer
PTRE	strcut Tree *	Tree pointer
PTSK	struct Task *	Task Pointer
PWTS	struct WindowsTask *	Windows Task Pointer
Radian	double	Radians

Name	Definition	Usage
Rad	Radian	Radians
Rct	struct Rect	Rect class
Real	double	Double precision number
Rea	double	Double precision number
SmallInt	char	Integers in range -127:+127
Sem	struct Semaphore	Semaphore class
Set	struct SetCollection	SetCollection class
Sor	struct SortArray	SortArray class
Stk	struct Stack	Stack class
String	char	Text data
Str	char	Text data
Tim	struct TimeData	TimeData class
TimeFormatCode	enum TimeFormatCode	Describe a display format for
Tms	struct TimeStampData	TimeStampData class
Tre	struct Tree	Tree class
True	((Bool)1)	Boolean True
Tsk	struct Task	Task class
TskThread	((PTSK)(TskList[*_threadid]))	Pointer to a threads Task object
ULargeInt	unsigned long	Integers in the range 0:0xFFFFFF
UMediumInt	unsigned int	Integers in the range 0:0xFFFF
USmallInt	unsigned char	Integers in the range 0:0xFF
Void	void	Function declarations
WtsThread	((PWTS)(WtsList[*_threadid]))	Pointer to a threads Window Task

C.2.2 Table B - Windowing Datatypes

Name	Definition	Usage
Aca	struct ApplCellArray	ApplCellArray class
Adt	struct ApplDesktop	ApplDesktop class
Aia	struct ApplItemArray	ApplItemArray class
Ama	struct ApplManagerArray	ApplManagerArray class
Asc	struct ApplStateCell	ApplStateCell class
Asi	struct ApplStateItem	ApplStateItem class
BitmapTypes	enum BitmapTypes	Bitmap types
Bmp	struct BitmapStorage	BitmapStorage class
Clb	struct ClipBoard	ClipBoard class
ClipboardFormats	enum ClipboardFormats	Clipboard formats
ControlEventType	enum ControlEventType	Control event types

Name	Definition	Usage
DataEntryType	enum DataEntryType	Validate date/time entry
Dde	struct DynamicDataExchange	DynamicDataExchange class
Dge	struct DialogEdit	DialogEdit class
DgeControlStyles	enum DgeControlStyles	Control edit types
Dgi	struct DialogItem	DialogItem class
DialogEventType	enum DialogEventType	Dialog event types
Dio	struct DialogItemIO	DialogItemIO class
Dvc	struct Device	Device class
Dvw	struct DialogEventWindow	DialogEventWindow class
Dwd	struct DialogBoxWindow	DialogBoxWindow class
Edt	struct Editor	Editor class
EditControlTypes	enum EditControlTypes	Edit control types
EditDataValidation	enum EditDataValidation	Edit Gdb return codes
Era	struct ErrorArray	ErrorArray class
Erd	struct ErrorDisplay	ErrorDisplay class
Eva	struct EventArray	EventArray class
Evw	struct EventWindow	EventWindow class
Fal	struct FontAvailabilityList	FontAvailabilityList class
Fmg	struct FontManager	FontManager class
Fnd	struct FontDialog	FontDialog class
Fon	struct FontElement	FontElement class
FrameEventType	enum FrameEventType	Frame window event types
Fse	struct FontSizeElement	FontSizeElement class
Fvw	struct FrameEventWindow	FrameEventWindow class
Fwd	struct FrameWindow	FrameWindow class
Gdb	struct GenericDialogBox	GenericDialogBox class
Glb	struct GraphicalListBox	GraphicalListBox class
Gps	struct GraphicsPresentation-Space	GraphicsPresentationSpace class
Hdl	struct HandleTracking	HandleTracking class
Lde	struct LineDisplayListElement	LineDisplayListElement class
Ldl	struct LineDisplayList	LineDisplayList class
Ldm	struct LineDisplayManager	LineDisplayManager class
Lre	struct LoadableResElement	LoadableResElement class
Lrs	struct LoadableResources	LoadableResources class
Mmg	struct MenuStateManager	MenuStateManager class
Mni	struct MenuItem	MenuItem class
Mnu	struct StandardMenu	StandardMenu class
Ofd	struct OpenFileDialog	OpenFileDialog class
OpenFileEventType	enum OpenFileEventType	Open file dialog event types

Name	Definition	Usage
PACA	struct ApplCellArray *	ApplCellArray pointer
PADT	struct ApplDesktop *	ApplDesktop pointer
PAIA	struct ApplItemArray *	ApplItemArray pointer
Pal	struct PaletteControl	PaletteControl class
PAMA	struct ApplManagerArray *	ApplManagerArray pointer
PASC	struct ApplStateCell *	ApplStateCell pointer
PASI	struct ApplStateItem *	ApplStateItem pointer
PBMP	struct BitmapStorage *	BitmapStorage pointer
PCLB	struct ClipBoard *	ClipBoard pointer
Pcf	struct PictureFrameControl	PictureFrameControl class
PDDE	struct DynamicDataExchange *	DynamicDataExchange pointer
PDGE	struct DialogEdit *	DialogEdit pointer
PDGI	struct DialogItem *	DialogItem pointer
PDIO	struct DialogItemIO *	DialogItemIO pointer
PDVC	struct Device *	Device pointer
PDVW	struct DialogEventWindow *	DialogEventWindow pointer
PDWD	struct DialogBoxWindow *	DialogBoxWindow pointer
PEDT	struct Editor *	Editor pointer
PERA	struct ErrorArray *	ErrorArray pointer
PERD	struct ErrorDisplay *	ErrorDisplay pointer
PEVA	struct EventArray *	EventArray pointer
PEVW	struct EventWindow *	EventWindow pointer
PFAL	struct FontAvailabilityList *	FontAvailabilityList pointer
PFMG	struct FontManager *	FontManager pointer
PFND	struct FontDialog *	FontDialog pointer
PFON	struct FontElement *	FontElement pointer
PFSE	struct FontSizeElement *	FontSizeElement pointer
PFVW	struct FrameEventWindow *	FrameEventWindow pointer
PFWD	struct FrameWindow *	FrameWindow pointer
PGDB	struct GenericDialogBox *	GenericDialogBox pointer
PGLB	struct GraphicalListBox *	GraphicalListBox pointer
PGPS	struct GraphicsPresentation-	GraphicsPresentationSpace
PHDL	struct HandleTracking *	HandleTracking pointer
PLDE	struct LineDisplayListElement *	LineDisplayListElement pointer
PLDL	struct LineDisplayList *	LineDisplayList pointer
PLDM	struct LineDisplayManager *	LineDisplayManager pointer
PLRE	struct LoadableResElement *	LoadableResElement pointer
PLRS	struct LoadableResources *	LoadableResources pointer
PMMG	struct MenuStateManager *	MenuStateManager pointer

Name	Definition	Usage
PMNI	struct MenuItem *	MenuItem pointer
PMNU	struct StandardMenu *	StandardMenu pointer
POFD	struct OpenFileDialog *	OpenFileDialog pointer
PPAL	struct PaletteControl *	PaletteControl pointer
PPCF	struct PictureFrameControl *	PictureFrameControl pointer
PPRE	struct PresentationParam *	PresentationParam pointer
PPRO	struct ProfileData *	ProfileData pointer
Pre	struct PresentationParam	PresentationParam class
Pro	struct ProfileData	ProfileData class
ProfileType	enum ProfileType	Profile types
PSAD	struct SaveAsDialog *	SaveAsDialog pointer
PSBW	struct ScrollBarWindow *	ScrollBarWindow pointer
PSCT	struct SelectedControl *	Selected Control pointer
PSLA	struct StatusLineArray *	StatusLineArray pointer
PSSA	struct StatusLineStateArray *	StatusLineStateArray pointer
PSTL	struct StatusLine *	StatusLine pointer
PSVW	struct ControlEventWindow *	ControlEventWindow pointer
PSWD	struct StandardWindow *	StandardWindow pointer
PTRT	struct TrackRectangle *	TrackRectangle pointer
PTBX	struct ToolBox *	ToolBox pointer
PTMA	struct ToolboxMenuStateArray *	ToolboxMenuStateArray pointer
PTSA	struct ToolboxStateArray *	ToolboxStateArray pointer
PWDG	struct WindowDialog *	WindowDialog pointer
PWIO	struct WindowInputOutput *	WindowInputOutput pointer
PWND	struct WindowBasic *	WindowBasic pointer
PWSA	struct WorkSpaceArray *	WorkSpaceArray pointer
ResourceTypes	enum ResourceTypes	Loadable resource types
SaveAsFile-EventType	enum SaveAsFileEventType	Save As file dialog event types
Sbw	struct ScrollBarWindow	ScrollBarWindow class
Sct	struct SelectedControl	Selected Control class
Sla	struct StatusLineArray	StatusLineArray class
Ssa	struct StatusLineStateArray	StatusLineStateArray class
Stl	struct StatusLine	StatusLine class
SysMenuConfig	enum SysMenuConfig	System menu configuartion
Svw	struct ControlEventWindow	ControlEventWindow class
Swd	struct StandardWindow	StandardWindow class
Trt	struct TrackRectangle	TrackRectangle class
Tbx	struct ToolBox	ToolBox class

Name	Definition	Usage
TimeStamp-Validation	enum TimeStampValidation-	Tms validation return codes
Tma	struct ToolboxMenuStateArray	ToolboxMenuStateArray class
ToolboxEventType	enum ToolboxEventType	Toolbox event types
Tsa	struct ToolboxStateArray	ToolboxStateArray class
Wdg	struct WindowDialog	WindowDialog class
WindowDlgEvent-Type	enum WindowDlgEventType	Window dialog event types
WindowDlgFormat	enum WindowDlgFormat	Window dialog transform formats
WindowEventType	enum WindowEventType	Window event types
Wio	struct WindowInputOutput	WindowInputOutput class
Wnd	struct WindowBasic	WindowBasic class
Wsa	struct WorkSpaceArray	WorkSpaceArray class

C.2.3 Table C - Database Access Datatypes

Name	Definition	Usage
Dbd	struct DatabaseDefinitions	DatabaseDefinitions class
Dcr	struct DlgControlDataRecord	DlgControlDataRecord class
Ddr	struct DialogDataRecord	DialogDataRecord class
Dge	struct DialogEdit	DialogEdit class
Fsq	struct FindSqlStatement	FindSqlStatement class
PDBD	struct DatabaseDefinitions *	DatabaseDefinitions pointer
PDCR	struct DlgControlDataRecord *	DlgControlDataRecord pointer
PDDR	struct DialogDataRecord *	DialogDataRecord pointer
PDGE	struct DialogEdit *	DialogEdit pointer
PFSQ	struct FindSqlStatement *	FindSqlStatement pointer
PSCL	struct SQLSystemColumns *	SQLSystemColumns pointer
PSTB	struct SQLSystemTables *	SQLSystemTables pointer
Scl	struct SQLSystemColumns	SQLSystemColumns class
SQLColumnTypes	enum SQLColumnTypes	SQL column types
Stb	struct SQLSystemTables	SQLSystemTables class

C.2.4 Table D - Parasol Kernel Datatypes

Name	Definition	Usage
Ab1	struct AbstractDataDialog1	AbstractDataDialog1 class
Abf	struct AbstractFindDialog	AbstractFindDialog class
Abl	struct AbstractListDialog	AbstractListDialog class
Aed	struct AbstractEditor	AbstractEditor class
Apf	struct ApplicationFindDialog	ApplicationFindDialog class
Apl	struct ApplicationListDialog	ApplicationListDialog class
Ap1	struct ApplicationDataDialog1	ApplicationDataDialog1 class
Bmd	struct BitmapDisplayDialog	BitmapDisplayDialog class
Bpf	struct BibProcedureFindDialog	BibProcedureFindDialog class
Bpl	struct BibProcedureListDialog	BibProcedureListDialog class
Bp1	struct BibProcedureDataDialog1	BibProcedureDataDialog1 class
Brd	struct BitmapRecordDialog	BitmapRecordDialog class
Ccd	struct ConeColorDialog	ConeColorDialog class
Ced	struct ConeEditor	ConeEditor class
Cev	struct ConeEditorEvents	ConeEditorEvents class
Cnd	struct ConnectionDialog	ConnectionDialog class
Cnl	struct ConnectionListDialog	ConnectionListDialog class
Cqe	struct ConnectionQueryElement	ConnectionQueryElement class
Dcd	struct DesignCardDialog	DesignCardDialog class
Ddm	struct DataDialogManager	Data Dialog Manager class
Dfd	struct DisplayFilterDialog	DisplayFilterDialog class
Dgr	struct DisplayGlobalRev	DisplayGlobalRev class
Dlg	struct DialogFunctions	DialogFunctions class
DlgColorsIndex	enum DlgColorsIndex	Dialog color type index
Dls	struct DisplayList	DisplayList class
Dob	struct DrawingObject	DrawingObject class
Dod	struct DesignOutlineDialog	DesignOutlineDialog class
Dpf	struct DialogColorPreference	DialogColorPreference class
Dsd	struct DisplaySettingsDialog	DisplaySettingsDialog class
Dsh	struct DisplayShapes	DisplayShapes class
Efd	struct EffectivityDialog	EffectivityDialog class
Eye	struct EyePerspectiveDialog	EyePerspectiveDialog class
Fde	struct FindDialogElement	FindDialogElement class
Fdm	struct FindDialogManager	FindDialogManager class
Flb	struct FindListBox	FindListBox class
Fma	struct FloatingMethodArray	FloatingMethodArray class
Geo	struct GeographicHelp	GeographicHelp class

Name	Definition	Usage
Hnp	struct HorizontalNodePoint	Horizontal or Edge Node class
Hom	struct SetHomeDialog	SetHomeDialog class
Itd	struct ItemDataDialog	ItemDataDialog class
Mcd	struct MenuCommands	MenuCommands class
Nld	struct NumberLevelsDialog	NumberLevelsDialog class
Nnp	struct ItemNodePoint	ItemNodePoint class
Oed	struct OutlineEditor	OutlineEditor class
Oev	struct OutlineEditorEvents	OutlineEditorEvents class
PAB1	struct AbstractDataDialog1 *	AbstractDataDialog1 pointer
PABF	struct AbstractFindDialog *	AbstractFindDialog pointer
PABL	struct AbstractListDialog *	AbstractListDialog pointer
PAED	struct AbstractEditor *	AbstractEditor pointer
PAPF	struct ApplicationFindDialog *	ApplicationFindDialog pointer
PAPL	struct ApplicationListDialog *	ApplicationListDialog pointer
PAP1	struct ApplicationDataDialog1 *	ApplicationDataDialog1 pointer
Par	struct ParStruct	ParStruct class
ParDesktopEvent-Type	enum ParDesktopEventType	Parasol desktop event types
Pdd	struct PageDesignDisplay	Page Design Dialog class
PBMD	struct BitmapDisplayDialog *	BitmapDisplayDialog pointer
PBPF	struct BibProcedureFindDialog *	BibProcedureFindDialog pointer
PBPL	struct BibProcedureListDialog *	BibProcedureListDialog pointer
PBP1	struct BibProcedureDataDialog1 *	BibProcedureDataDialog1 pointer
PBRD	struct BitmapRecordDialog *	BitmapRecordDialog pointer
PCCD	struct ConeColorDialog *	ConeColorDialog pointer
PCED	struct ConeEditor *	ConeEditor pointer
PCEV	struct ConeEditorEvents *	ConeEditorEvents pointer
PCND	struct ConnectionDialog *	ConnectionDialog pointer
PCNL	struct ConnectionListDialog *	ConnectionListDialog pointer
PCQE	struct ConnectionQueryElement *	ConnectionQueryElement pointer
PDCD	struct DesignCardDialog *	DesignCardDialog pointer
PDDM	struct DataDialogManager *	Data Dialog Manager pointer
PDFD	struct DisplayFilterDialog *	DisplayFilterDialog pointer
PDGR	struct DisplayGlobalRev *	DisplayGlobalRev pointer
PDLG	struct DialogFunctions *	DialogFunctions pointer
PDLS	struct DisplayList *	DisplayList pointer
PDOB	struct DrawingObject *	DrawingObject pointer
PDOD	struct DesignOutlineDialog *	DesignOutlineDialog pointer
PDPF	struct DialogColorPreference *	DialogColorPreference pointer

Name	Definition	Usage
PDSD	struct DisplaySettingsDialog *	DisplaySettingsDialog pointer
PDSH	struct DisplayShapes *	DisplayShapes pointer
PEFD	struct EffectivityDialog *	EffectivityDialog pointer
PEYE	struct EyePerspectiveDialog *	EyePerspectiveDialog pointer
PFDE	struct FindDialogElement *	FindDialogElement pointer
PFDM	struct FindDialogManager *	FindDialogManager pointer
PFLB	struct FindListBox *	FindListBox pointer
PFMA	struct FloatingMethodArray *	FloatingMethodArray pointer
PGEO	struct GeographicHelp *	GeographicHelp pointer
PHNP	struct HorizontalNodePoint *	Horizontal or Edge Node pointer
PHOM	struct SetHomeDialog *	SetHomeDialog pointer
PITD	struct ItemDataDialog *	ItemDataDialog pointer
PMCD	struct MenuCommands *	MenuCommands pointer
PNLD	struct NumberLevelsDialog *	NumberLevelsDialog pointer
PNNP	struct ItemNodePoint *	ItemNodePoint pointer
POED	struct OutlineEditor *	OutlineEditor pointer
POEV	struct OutlineEditorEvents *	OutlineEditorEvents pointer
PPAR	struct ParStruct *	ParStruct pointer
PPDD	struct PageDesignDisplay *	Page Design Dialog pointer
PPMG	struct ParMessageData *	ParMessageData pointer
PPPV	struct PrintPreviewDialog *	PrintPreviewDialog pointer
Ppv	struct PrintPreviewDialog	PrintPreviewDialog class
PPSD	struct PrintStartDialog *	PrintStartDialog pointer
PPSL	struct PrinterSelectList *	PrinterSelectList pointer
PQEM	struct QueryStackManager *	QueryStackManager pointer
PRVF	struct RevisionFindDialog *	RevisionFindDialog pointer
PRVL	struct RevisionListDialog *	RevisionListDialog pointer
PRV1	struct RevisionDataDialog1 *	RevisionDataDialog1 pointer
Psd	struct PrintStartDialog	PrintStartDialog class
PSGR	struct SetGlobalRev *	SetGlobalRev pointer
PSIA	struct SelectedItemArray *	SelectedItemArray pointer
PSIE	struct SelectedItemElement *	SelectedItemElement pointer
Psl	struct PrinterSelectList	PrinterSelectList class
PSLD	struct SelectionDataDialog *	SelectionDataDialog pointer
PSTS	struct AppStatusLine *	AppStatusLine pointer
PT3D	struct Transforms3D *	Transforms3D pointer
PTMG	struct TreeManager *	TreeManager pointer
Qem	struct QueryStackManager	QueryStackManager class
Rvf	struct RevisionFindDialog	RevisionFindDialog class

Name	Definition	Usage
Rvl	struct RevisionListDialog	RevisionListDialog class
Rv1	struct RevisionDataDialog1	RevisionDataDialog1 class
Sgr	struct SetGlobalRev	SetGlobalRev class
Sia	struct SelectedItemArray	SelectedItemArray class
Sie	struct SelectedItemElement	SelectedItemElement class
Sld	struct SelectionDataDialog	SelectionDataDialog class
Sts	struct AppStatusLine	AppStatusLine class
T3d	struct Transforms3D	Transforms3D class
Tmg	struct TreeManager	TreeManager class

C.2.5 Table E - Parasol SQL Interface Database Datatypes

Name	Definition	Usage
Asr	struct AbstractRecord	AbstractRecord class
Bmr	struct BitmapRecord	BitmapRecord class
Bpr	struct BibProcedureRecord	BibProcedureRecord class
Cnc	struct ConnectionData	ConnectionData class
Cxi	struct ConnectionExplImpl	ConnectionExplImpl class
Grv	struct GlobalRevisionData	GlobalRevisionData class
Hrf	struct HelprefData	HelprefData class
Itm	struct ItemData	ItemData class
Itt	struct ItemtypeRecord	ItemtypeRecord class
PASR	struct AbstractRecord *	AbstractRecord pointer
PBMR	struct BitmapRecord *	BitmapRecord pointer
PBPR	struct BibProcedureRecord *	BibProcedureRecord pointer
PCNC	struct ConnectionData *	ConnectionData pointer
Pcr	struct ParcontrolDataRLO	ParcontrolDataRLO class
PCXI	struct ConnectionExplImpl *	ConnectionExplImpl pointer
PGRV	struct GlobalRevisionData *	GlobalRevisionData pointer
PHRF	struct HelprefData *	HelprefData pointer
PITM	struct ItemData *	ItemData pointer
PITT	struct ItemtypeRecord *	ItemtypeRecord pointer
Pmg	struct ParMessageData	ParMessageData class
PPCR	struct ParcontrolDataRLO *	ParcontrolDataRLO pointer
PPRM	struct ParasolRecordManager *	ParasolRecordManager pointer

Name	Definition	Usage
PREV	struct RevisionRecord *	RevisionRecord pointer
Prm	struct ParasolRecordManager	ParasolRecordManager class
PSLR	struct SelectionDataRLO *	SelectionDataRLO pointer
Rev	struct RevisionRecord	RevisionRecord class
Slr	struct SelectionDataRLO	SelectionDataRLO class
ParMessage-	enum ParMessageType	Parasol event types
ITT_UNKNOWN	#define 0	Definition of unknown table type
ITT_ABSTRACT	#define 1	Definition of abstract table type
ITT_BIBPROC	#define 2	Definition of procedure table type

- PARENTID - Parent record key
- COMPONENTTYPE - Child record table type. This is the number defined in the ITEMTYPE table TYPE field.
- COMPONENTID - Child record key
- FINDNBR - {Reserved}
- REVLEVELIN - Revision level of the in effective connection
- TYPE - Type of connection
- EFFECTDATEIN- Date the connection is in effective